DOCTOR OF ENGINEERING (ENGD)

A Novel Neural Network Architecture with Applications to 3D Animation and Interaction in Virtual Reality

Dehesa, Javier

Award date:
2021

Awarding institution:
University of Bath

A Novel Neural Network Architecture with Applications to 3D Animation and Interaction in Virtual Reality

submitted by

Javier de la Dehesa Cueto–Felgueroso

for the degree of Doctor of Engineering

of the

University of Bath

Department of Computer Science

November 2020

Abstract

Realistic real-time character animation in 3D virtual environments is a difficult task, especially as graphics fidelity and production standards continue to rise in the industry. Motion capture is an essential tool to produce lifelike animation, but by itself it does not solve the complexities inherent to real-time environments. In the case of interactive characters in virtual reality, this difficulty is taken to another level, as the animation must dynamically adapt to the free actions of the user. On top of that, these actions, unlike the touch of a button, are not easily interpretable, introducing new challenges to interaction design.

We propose a novel data-driven approach to these problems that takes most of this complexity out of the hands of the developers and into automated machine learning methods. We propose a new neural network architecture, "grid-functioned neural networks" (GFNN), that is particularly well suited to model animation problems. Unlike previous proposals, GFNN features a grid of expert parameterisations associated with specific regions of a multidimensional domain, making it more capable of learning local patterns than conventional models. We give a full mathematical characterisation of this architecture as well as practical applications and results, evaluating its benefits as compared with other state-of-the-art models. We then propose a complete framework for human–character interaction in virtual reality built upon this model, along with gesture recognition and behaviour planning models. The framework establishes a novel general data-driven approach to the problem applicable to a variety of scenarios, as opposed to existing ad hoc solutions to specific cases. This is described at an abstract, general level and in the context of a particular case study, namely virtual sword fighting, demonstrating the practical implementation of these ideas.

Our results show that grid-functioned neural networks surpass other comparable models in aspects like control accuracy, predictability and computational performance, while the evaluation of our interaction framework case study situates it as a strong alternative to traditional animation and interaction development techniques. This contributes to the growing trend of incorporating data-driven systems into video games and other interactive media, which we foresee as a convergent future for industry and academia.

Acknowledgements

Even though there is only one name under the title of this thesis, the truth is that very little would be found within its covers were it not for the of support many more people and organisations to whom I shall here express my gratitude.

To my supervisory team, Julian Padget, Christof Lutteroth and Andrew Vidler, who have guided my every step for the last four years. Your advice, your feedback, your ideas and your corrections are the tools with which I have built this work. I could not have asked for a more brilliant and supportive team to accompany me in this journey.

To the University of Bath, and especially to the Centre for Digital Entertainment, that gave me the opportunity to get involved in this in the first place. You have really given the best of yourselves to make sure we students could always count with your help, whatever concerns, doubts or problems came our way.

To Ninja Theory, both as a company and as a team of incredibly talented individuals. You have welcomed me like one more ninja since the very first day, and the trust and support that you have given me has been a constant motivation. I feel enormously privileged for the time I shared with all of you and everything you have taught me.

To the Engineering and Physical Sciences Research Council, which has funded my endeavours throughout these years.[1] Let this work be a humble exponent of publicly-funded research as the staple of human knowledge advancement and societal progress.

To my loving wife Karem, the rock under my feet, ever supporting, ever encouraging. You have been with me through the best and the worst of times, and there is no doubt I would not have made it to even half of the way without you. You are the most faithful, devoted and patient partner that I could ever wish for.

To my parents, siblings and extended family, friends and acquaintances, and everyone who has been rooting for me in this enterprise. I see you, I cherish you, and I hope you all feel as loved as I did in all your future undertakings.

To all of you, from the bottom of my heart, thank you.

[1] EPSRC grant reference EP/L016540/1.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

As the digital entertainment industry continues to advance and evolve, the size, depth and realism of the media it produces keep growing on an unprecedented scale. In the case of real-time interactive digital experiences, meaning, for the most part, video games, this progress is even more impressive considering the inherent dynamism of the medium, its performance requirements and the limitations of consumer hardware. The development of large-scale video games with high production values is complex and laborious, requiring vast amounts of digital assets of all kinds, like character models, environmental objects, textures or audio tracks. It is not surprising that a large share of the research output in the field addresses the design of tools and techniques that assist or simplify the production of novel content, when not outright generating it themselves.

Animation is one of the areas where the need for better computer-assisted content creation tools is most pressing. As visual fidelity in 3D graphics evolves, the human eye demands ever more accurate and organic animation in order to escape the infamous "uncanny valley" (Mori, MacDorman and Kageki, 2012). Motion capture (fig. 1-1), now a standard in high-end video game productions, is an invaluable tool towards this goal, but not sufficient on its own. The myriad of scenarios, interactions and situations present in current video games can make the design of a realistic and reliable animation system a gargantuan task even for a professional animator. For example, a simple locomotion action can be influenced by factors as diverse as the walking speed, the shape of the terrain, the environmental conditions, the surrounding objects or the mood of the character. Animation logic must be able to combine all these variables to produce the right kind of motion, all while respecting the control directives from the user.

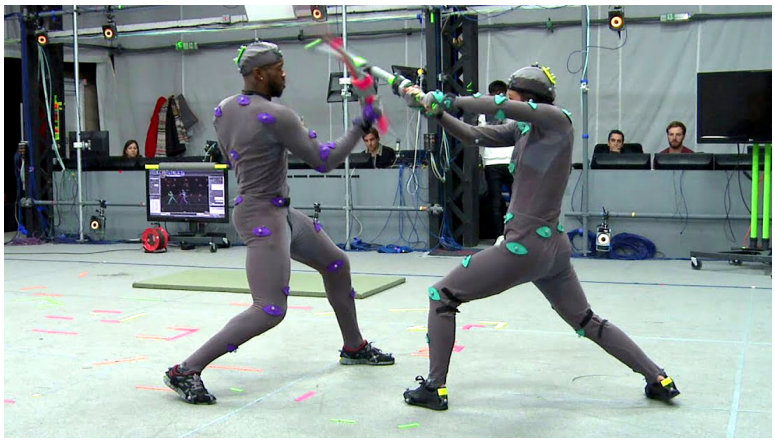


Figure 1-1: Motion capture is commonly used in modern video games to bring a character's body and face to life (Ninja Theory, 2016).

Virtual reality (VR), which grows more and more important as technology becomes established, makes this problem even more complicated. It offers the user the possibility to interact with the environment, and therefore with virtual characters, through free hand motions, not restricted to any strict pattern or sequence. This new form of interaction does not only mean that the animation must adapt to this unpredictable input, but also that the interpretation of the actions from the user is no longer obvious, turning it into a new kind of difficulty. Far from stabilising, the scale of these issues is expected to keep increasing as productions become even more sophisticated and complex. It is now the time for new approaches and methods that take full advantage of the computational power we have today to radically facilitate the work of animators and developers, and data-driven solutions are one of the most promising routes to that end.

Data-driven methods have been used in real-time animation for a long time. As mentioned above, motion capture is a foundational technology to replace hand-authored animation. But more recently this perspective has been taken further in several directions with the aim of completely automating the way in which animation data is selected and combined. Machine learning, and in particular neural networks, has delivered a number of impressive results in this field so far. Unfortunately, existing models do not always satisfy all the needs required by more complex or specialised scenarios, presenting important limitations that restrict their more general application. What is more, these are exclusively focused on the animation problem itself, but do not take into consideration how this animation becomes affected by the larger context, which is a fundamental issue in VR. We need a more general and comprehensive approach that

provides developers with a complete framework to build data-driven VR experiences.

1.1 Research Goals

This thesis delves specifically into two distinct topics. The first one is the problem of data-driven animation synthesis on its own, studied from the perspective of neural networks. In particular, we investigate the generation of realistic real-time animation from motion capture data. The difficulty of this problem resides in the fact that the model must be able to learn the whole domain of motion attributable to the character, without simplifications. Taking the example of locomotion, the feet of the character may adopt a variety of configurations, and the neural network must be able to reproduce them all. This, which may seem obvious, is actually difficult because the trained model will be strongly drawn towards smoothed results that fail to synthesise the distinct motion involved in each of the actions with sufficient detail. While several proposals have been put forward in this area, inherent issues in their conception hinder their usability and performance. Our first goal is therefore to come up with a new approach to modelling this kind of problem that is more adaptable to new scenarios and offers a better compromise between accuracy, computational cost and reliability.

The second topic addresses the larger problem of data-driven human–character interaction in VR. What is meant by this is the utilisation of data-driven models to facilitate the creation of interactive scenarios in VR with animated characters. This topic includes, in a sense, the previous one, as any virtual character involved in the interaction will require some method of animation generation. But, furthermore, it also takes into account the question of how to interpret the actions of a user in VR. This is an issue that is intrinsic to VR as a medium, where the traditional controller is replaced by more natural motion-tracked control. Interpreting the free actions of a user becomes then a challenge on its own, which must be resolved as a precondition to any interaction. On top of that, adherence of the character to the prescribed behaviour is paramount, which in practice restricts the scope of application of purely data-driven methods. There is little prior work considering this problem as a whole, and only in the form of specific solutions to specific scenarios. A more general methodology would provide a proven foundation to approach the development of any new scenario. Thus, our goal here is to define a theoretical framework of common constructs that can be generally applied to the design of interactive VR experiences, combining data-driven models and human expertise in a systematic and predictable manner.

In summary, our research aims to answer the following questions: how can a neural

network reliably learn all the variety in a character's animation? And how can we use such a neural network, along with other models, to build interactive experiences in VR more productively and with higher quality?

1.2 Contributions

In accordance to the goals outlined above, we present the following contributions, developed throughout the following chapters of this work:

- A new kind of neural network architecture called "grid-functioned neural networks" capable of breaking down a problem across multiple dimensions and learning local patterns that improve the overall performance of the model. This is a general architecture applicable to any problem where regular neural networks can be used.
- A study of the benefits of this architecture applied to the problem of quadruped locomotion animation, and specifically as compared to another state-of-the-art model, considering not just the accuracy of the results but also the computational performance and user perception.
- A general framework for data-driven interaction in VR that makes use of our proposed neural network architecture, along with neural gesture recognition and handcrafted behaviour design, to automate the lower-level tasks of the development while keeping control of the design at general level.
- A complete case study of this framework applied to a VR sword fighting scenario, including the implementation of each of the steps and components in our proposed methodology and its evaluation from numerical and user-centred perspectives.

These contributions complete a novel data-driven approach to character design that puts machine learning at the service of animators and developers. Our animation and interaction models work as a support in the process of bringing new VR experiences to reality, simplifying the production cycle while keeping true to the designer's vision.

1.3 Industrial Context

It is worth noting that the presented research is the fruit of a partnership between the University of Bath and the video games studio Ninja Theory Ltd. Ninja Theory is an award-winning studio based in Cambridge, UK, that specialises in story-rich games with high production values. It has a strong research and development arm, as demonstrated by the innovations in real-time motion capture (fig. 1-2) and binaural sound rendering that took part in the production of their title *Hellblade: Senua's Sacrifice* (Antoniades

Figure 1-2: Ninja Theory earned recognition from the computer graphics community for their innovations in real-time cinematography.

et al., 2016). As future projects grow more ambitious, Ninja Theory is ever more invested in the adoption of smart technology that empowers and boosts the productivity of artists and developers. Our VR interaction framework leverages recent advances in machine learning to achieve this precise goal.

Even though the contributions introduced in this work stand on their own as purely academic outputs, they are imbued with an industrial-level quality provided by the environment where they have been developed. This means these are not just merely theoretical or abstract proposals, but they are endorsed by industrial experts as practical tools and methods to be eventually integrated into a real development workflow. While no commercial works feature this technology yet, our results have already impacted the approach that Ninja Theory is taking to animation and interaction in upcoming titles, embracing a future where data-driven methods become a standard in the production of highly realistic video games.

1.4 Associated Publications

The work herein presented was first partially introduced in the publications "Towards Data-Driven Sword Fighting Experiences in VR" (Dehesa, Vidler et al., 2019) and "Touché: Data-Driven Interactive Sword Fighting in Virtual Reality" (Dehesa, Vidler et al., 2020). The contributions in these publications are, for the most part, discussed in chapter 5, though results from chapter 3 are included as well.

1.5 Document Structure

The rest of this thesis is structured as follows. Chapter 2 reviews prior literature related to our work, surveying both comparable and alternative approaches to the problems under discussion. Chapter 3 details the mathematical definition of grid-functioned neural networks as a general machine learning model, including concrete results on a set of synthetic regression problems that highlight its strengths. Chapter 4 then moves on to the specifics of animation, demonstrating the performance of grid-functioned neural networks on the problem of quadruped locomotion animation synthesis and showing its advantages over the prior state of the art. Finally, chapter 5 introduces the complete framework for data-driven interaction in VR, both at a conceptual level and in the form of an extensively evaluated case study. The thesis concludes with a discussion on the presented work and final thoughts in chapter 6.

Chapter 2

Background and Previous Work

Real-time animation and interaction in VR are broad topics reaching a variety of areas of knowledge. This chapter reviews works in some of these areas that are relevant to the framework we are presenting. First, we situate the present work with a short survey of several fields that are key to the framework introduced in chapter 5, starting with virtual humans as a general concept, then animation synthesis, followed by gesture recognition and finally artificial reasoning. We then include a more detailed discussion on neural networks as a general tool and on phase-functioned neural networks, which will lay the mathematical foundations for our exposition in chapter 3.

2.1 Virtual Humans

Researchers of all sorts of branches of computer science have always been fascinated by the possibility of creating virtual human-like characters, capable of interacting with real users in a natural fashion. It is not only an interesting topic in itself, there is an abundance of potential applications and evidence suggesting that virtual humans would significantly improve human-computer experiences where actual human interaction is not available. And while we are still far from full-fledged virtual humans, we do not need to reach that point in order to incite a believable sense of interaction in users. For example, it has been repeatedly noted that proxemic behaviour, which regulates the interpersonal distance between subjects, is similarly affected by virtual characters in immersive environments as it would be by humans in the real world (Bailenson et al., 2003; Llobera et al., 2010; Kolkmeier, Vroon and Heylen, 2016). And, in fact, immersive interaction with virtual characters has already shown potential for therapeutic (Kandalaft et al., 2013), educative (Fitton, Finnegan and Proulx, 2020) and artistic

applications (Batras et al., 2016).

We can find an early example of a virtual assistant in the work of Rickel and Johnson, (1999), oriented towards operational training. The user, situated in a virtual environment with a head-mounted display and a data glove, could "touch" the surrounding objects and interact with the assistant through speech, which in turn would issue instructions and explanations accordingly. The agent displayed fairly advanced capabilities, featuring a rich cognition subsystem based on Soar, a comprehensive architecture for the development of intelligent agents (Laird, Newell and Rosenbloom, 1987). Overall, it was quite a flexible system, but obviously very dated now in visual terms. For instance, the animation of the assistant was limited to gaze direction and pointing, without any direct interaction with the user.

In a different domain, Noma, L. Zhao and Badler, (2000) developed a virtual character for public presentations. Their work combined several basic elements to produce feature-rich presentations, even though it makes no attempt at modelling cognition or natural behaviour. Instead, the agent was provided with a script including text and gestural indications, that would then be enacted by the character using a virtual board and other objects. Using an ensemble of finite automata, the virtual presenter can perform multiple simultaneous task while body motion is synchronised with the synthesised speech. Their model produced real-time animations, but it did not allow for interaction from the audience.

Looking into more recent developments, SimSensei Kiosk, by DeVault et al., (2014), is a very interesting exploration of the possibilities of virtual humans in the field of mental wellbeing. It presented a feature-rich virtual interlocutor designed to assess distress indicators related to different psychological disorders. The underlying system, tailored to the specific application, includes multiple nonverbal communication perception mechanisms (facial expression, gaze direction, etc.), speech recognition with basic natural language understanding and empathetic body language. It is a notable example of a reasoning agent featuring reactive animation, even though it is not presented within a fully immersive environment. Also, being focused on the communication with the user, it does not model more intricate spatial interactions.

The team at the Laboratory for Animate Technologies at the University of Auckland has been working on realistic recreations of virtual humans for several years, and much of their work can be found condensed into BabyX (Sagar, 2015). This is a virtual baby capable of reacting to visual input from humans with facial animation and imitation learning, among other things. Interestingly, they strive to recreate biologically plausible

animation and mind modelling. It is also one of the most visually advanced academic projects in the area, exploiting modern computer graphics hardware and techniques to its fullest. As of now, however, they have not investigated the possibility of direct interaction of the character with objects and humans in a virtual world.

As these works show, research in virtual humans has been typically focused on the cognitive aspects of the character, such as being able to emulate some particular aspect of human behaviour or help a user towards some goal. However, they generally do not study complex interactions with humans and, in particular, they are not designed for interactive VR environments. Also, as they are very application-specific agents, their behavioural features are hand-built with very precise goals. They do not contemplate the use of data-driven methods to address looser behaviour specifications more effectively. By contrast, our interaction model is not centred around the emulation of any particular cognitive task, but it can be flexibly adapted to multiple scenarios as long as the necessary data is available.

2.2 Character Animation Synthesis

Real-time character animation remains a notorious challenge in high-quality digital productions. The advent of mature motion capture technology in the 1990s opened the door to unprecedented realism in character movement, but the new difficulties it posed, especially for interactive media, soon became evident. Effective manipulation techniques for motion capture data have been studied for a long time now (Bruderlin and L. Williams, 1995; Witkin and Popovic, 1995; Gleicher, 2001), but producing a continuous stream of fluid motion from a collection of motion clips in real time involves a new level of complexity, since it requires the generation of natural transitions between independent animation sequences in real time. Perlin, (1995) outlined an early proposal for an animation framework based on mixtures or blends of multiple hand-crafted actions. Today, the most common approach to the problem is to manually design complicated state machines and "blend trees" expressing interpolation of multiple clips according to different combinations of user input and environmental information (Mizuguchi, Buchanan and Calvert, 2001). While this method allows for great control and predictability, it is very time-consuming and difficult to scale, and the results are hardly reusable at all.

An influential concept in automatic locomotion animation is the motion graph structure introduced by Kovar, Gleicher and Pighin, (2002), a graph storing short animation clips in the edges such that any path within it can be mapped into a smooth animation. Motion graphs can be constructed automatically from motion capture data used to

animate a user-controlled character. This model has been used as a basis for different extensions (Kovar and Gleicher, 2004; Heck and Gleicher, 2007) and as inspiration for entirely new designs. Treuille, Y. Lee and Popović, (2007) proposed a comparable approach for bipedal motion using control theory, splitting the data in individual step cycles that define, implicitly, a trivial motion graph, and a series of approximations to the optimal planning are devised. These works shaped the design of the crowd animation system in the video game *Hitman: Absolution* by IO Interactive (Büttner, 2013). A further development of the idea was the introduction of motion fields by Y. Lee et al., (2010), which can be roughly described as a continuous version of a motion graph. In this case animations live in a high dimensional vector space embedding character poses, and every frame is dynamically evaluated using an approximated policy reward function. Ubisoft Montreal developed a simplified version of motion fields named motion matching to animate the characters of the video game *For Honor* (Clavet, 2016).

The aforementioned systems, however, are all non-parametric models, meaning that their complexity increases with the size of the data, so they are in general not scalable to arbitrarily rich characters. Neural networks have been found to be a plausible parametric alternative, featuring favourable traits in terms of memory and computational cost. Taylor, Hinton and Roweis, (2007) and Fragkiadaki et al., (2015) showed different neural models able to apprehend the patterns in a motion capture database and generate streams of animation in real time. Recently, Holden, Komura and Saito, (2017) designed an interactive animation system using a phase-functioned neural network architecture, replacing the usual weights of the neural network with a multi-dimensional closed spline evaluated at a "phase" point. The phase represents the location of the current pose in a walking cycle, so the proposal is specifically oriented to locomotion. The system is able to reproduce realistic animations on a variety of environments with multiple styles. Its success led to derivative works that take the idea beyond its initial scope, like the mode-adaptive neural network (Zhang et al., 2018), neural state machine models (Starke, Zhang et al., 2019) and further iterations on the same idea (Starke, Y. Zhao et al., 2020), which replace the phase with another "gating" neural network. Our grid-functioned neural network architecture introduced in chapter 3 is also built on the principles of phase-functioned neural networks, and a detailed description of the model is included at the end of this chapter. Mode-adaptive neural networks are also discussed further in chapters 3 and 4. On the other hand, Holden, Kanoun et al., (2020) also proposed an alternative implementation of the concepts of motion matching that replaces the large animation databases in the original design with neural networks, resulting in a vast reduction of memory requirements at the expense of higher computational cost and slightly less fidelity.

One distinct variant of the problem of character animation is that including interactions between a human and a character in proximity, particularly relevant for our goals, for which several approaches have been proposed. J. Lee and K.H. Lee, (2006) used unlabelled motion capture data to produce control policies for characters in dynamic environments. Using a reinforcement learning approach, the authors are able to emulate boxing or tennis playing scenarios, although real-time direct interaction with users is not really explored. Ho and Komura, (2011) proposed a topological characterisation of wrestling animation for two virtual agents, allowing a user to interactively decide the actions of one of the characters. However the model is not necessarily applicable to other interactions, nor it is suited to human motion input. Ho, Chan et al., (2013) outline a more general method based for human–character interaction animation. The method queries a captured motion database with the current state of the user and character to search for similar recorded clips. Using animation editing techniques, the retrieved motion is morphed to match the actual situation. The method is demonstrated with close dance interactions in VR. Using a statistical approach, Taubert et al., (2013) constructed a hierarchical model for real-time interactions between a human and a virtual character. The authors encoded additional variables in the model to represent emotional expressions and space locations, showcased in a character able to "high-five" a user in a virtual environment with different styles. Vogt et al., (2014) presents a data-driven method conceptually similar to Ho, Chan et al., (2013), but with several improvements, including a better low-dimensional representation of the current queried pose and both spatial and temporal matching with the stored data. Scenarios featuring "high five" and clapping game interactions are emulated using the method. These techniques solve the problem at hand to a certain degree but, being completely data-driven, they cannot be directly controlled through custom logic in order to display specific behaviours, as we require.

Finally, it is worth discussing recent advances in physics-based animation synthesis using reinforcement learning. This is a fairly different domain where, instead of emulating existing motion capture data, the character controller learns to perform tasks by means of rewards and penalties that inform the training of the model. This is a very flexible approach, as demonstrated in the work by Heess et al., (2017), where not only a humanoid character learns to walk and avoid obstacles through this mechanism, but other kind of skeletal figures too. However, pure reinforcement learning rarely produces organic, human-like animation by itself, and the quality of the results heavily depends on hand-tuning of the physical model of the character and the design of the rewards. Several authors have proposed hybrid methods that actually use motion data to bootstrap the controller to an acceptable state, which then is further trained through reinforcement

learning to implement character control (Bergamin et al., 2019; Ling et al., 2020; Luo et al., 2020). However, these still require a computational model of the environment itself in order to be trained, which is not always easy to construct, and in the case of human–character interaction it may be simply unfeasible.

Character animation synthesis is one of the main topics of the work presented throughout the following chapters. Our approach to the problem is based on a particular kind of neural network inspired by the phase-functioned neural network architecture, which shows very successful control capabilities in complex scenarios. Chapter 3 introduces this model in abstract terms, while chapter 4 describes its application to a specific animation problem.

2.3 Gesture Recognition

As mentioned in chapter 1, gesture recognition will play an important role in our VR interaction framework. This is a topic that encompasses a varied set of problems that have been subject to study for a long time now. The extensive survey by Mitra and Acharya, (2007), even if not up to date with the latest developments, does a good job characterising and categorising the variety of problems and approaches proposed within the field. We can classify a gesture recognition scenario in terms of the kind of gestures under consideration, the nature of the available data and the time at which the prediction is yielded. Since our focus is on VR interaction, we will be looking at sensor information, coming from the VR controllers, which must be processed in real time. In fact, we will be interested not only in detecting gestures as soon as they are completed, but also in having an early recognition as soon as a gesture begins, and an indication of the progression of the gesture through time. As will be explained in chapter 5, this characteristic is used by our framework to inform the behaviour of the virtual character. In terms of gesture types, as most commercially available hardware does not yet feature finger-tracked control, the scope shall be limited to broad arm-level gestures, like waving or clapping.

Arguably, the most frequently used model in gesture recognition is the Hidden Markov Model (HMM), as first proposed by Yamato, Ohya and Ishii, (1992). There is good reason for that; it is a very well-known method, it has demonstrated success in a multitude of contexts and, importantly, it works with any kind of feature vector extracted from the input. This means it is a very flexible framework that can be used (and has been used) with virtually any kind of data. HMMs, as a statistical model (Rabiner and Juang, 1986), are a powerful technique for pattern recognition in sequential data. An

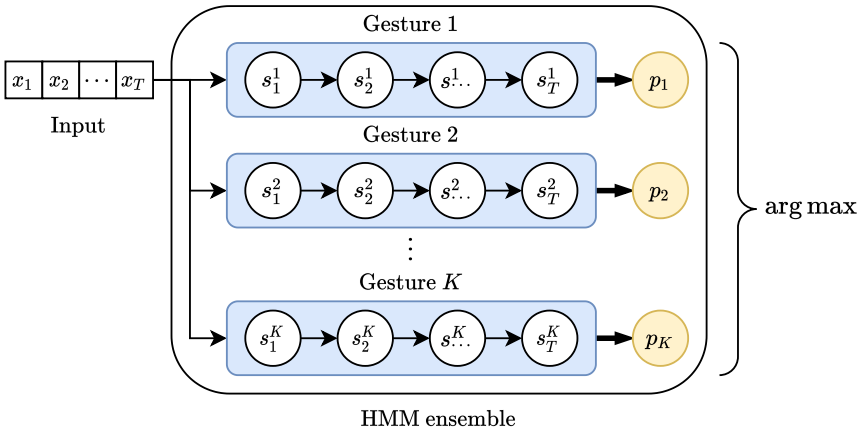


Figure 2-1: Hidden Markov Model (HMM) for gesture recognition.

HMM defines a finite number of hidden states, each of which is associated with a transition probability to every other state and an output probability distribution. Given an initial state and a sequence of data, it is possible to estimate the likelihood that the sequence was generated by the recurrent process of sampling a value from the current state and transitioning to another state. This can be used to classify gestural sequences as shown in fig. 2-1. For each gesture, an HMM is fitted to maximise the likelihood of the available examples. Given a new sequence $\boldsymbol{x}_{1\ldots T}$, for each gesture i the most likely sequence of hidden states $s_1^i, \ldots, s_T^i$ is estimated, along with a likelihood p_i. The model yielding the highest likelihood determines then the predicted gesture class. This is a fairly straightforward scheme, but there are several obvious limitations to it. First, this model assumes segmented gestural data as input. If gestures just happen spontaneously within a stream of data (as is the case for a VR controller), there is no general method to tell what slices of data should be fed to the model. And even if the bounds of each gesture are known, the recognition is only given at its end, with no information during its progression. Also, HMMs do not inherently handle the case where a given sequence does not belong to any of the known classes (e.g. because it is not a gesture or is a gesture unknown to the system).

Several authors have put forward different solutions to these problems. Sometimes, the system simply can be designed so the user may signal the gesture boundaries in a natural way (Kallio, Kela and Mäntyjärvi, 2003; Mäntyjärvi et al., 2004; Elmezain et al., 2008; Schlömer et al., 2008). In VR, however, expecting users to press a button or perform some other action along with each gesture is likely to degrade the sense of immersion of the user, which is the main feature of the medium. Different heuristics to detect the endpoints of a gesture from changes in the data have been successful in some contexts

(Liang and Ouhyoung, 1998; Kim et al., 2009; Raffa et al., 2010). Unfortunately, it is not easy to design such a method for problems that may include two-handed gestures or a combination of static gestures (poses) and dynamic ones. There are also proper extensions to the HMM model, such as that proposed by Yin and Davis, (2014), who used a hierarchical model to support continuous gesture recognition with a small delay for both static and dynamic gestures, even providing basic information about the progression of the gesture. Other variations include hidden conditional random fields (Wang et al., 2006), which combine class-based HMMs into a unified probabilistic model, or hidden conditional neural fields (Lu, Tong and Chu, 2016), that take this idea further, adding a "gating" step akin to a neural network layer to the model. These do improve the performance of the method, but do not add new features in terms of real-time behaviour, meaning they cannot detect gesture boundaries or recognise gestures as they are happening.

HMMs are not the only temporal model suitable for gesture recognition. Techniques like dynamic time warping (DTW) have been successfully applied in the works by Keskin, Cemgil and Akarun, (2011) and Reyes, Domínguez and Escalera, (2011). This algorithm measures the similarity between two sequences of data in terms of the changes that would be necessary to match them. This can be used for pattern identification, but DTW cannot in general be used over arbitrarily long streams with multiple gestures, since it is designed to match a single pattern to a single input sequence. Also, like other approaches, its original purpose is the analysis of sequences of already completed gestures. A more extensive review of this and other time-based approaches can be found in the survey by Cheng, Yang and Liu, (2016).

On the other hand, more general-purpose methods have also been applied to gesture recognition. Neural networks have had sporadic appearances in the literature for a long time (Weissmann and Salomon, 1999; Xu, 2006), but more recently they have overtaken most of the space in visual gesture recognition (Escalera et al., 2014; Neverova et al., 2014; Molchanov et al., 2016; Tsironi, Barros and Wermter, 2016; Asadi-Aghbolaghi et al., 2017). Like in most areas of computer vision, convolutional networks quickly emerged as the new state of the art in this area in the last decade, although other models, like support vector machines (which benefit from low memory and computational requirements), are still used with very competitive performance (Fanello et al., 2017). Some of these computer vision works deal specifically with the problem of early action recognition. Kong, Kit and Fu, (2014) proposed a multiple temporal scale support vector machine with predictors for different stages of an action. Their model does not solve the problem of identifying action boundaries though, and it is not clear how the

different temporal granularity levels would be defined for an online system. Different combinations of convolutional and recurrent layers have also been investigated to this end (Ma, Sigal and Sclaroff, 2016; Kong, Gao et al., 2018), although their ability to perform early detection and progress prediction throughout a long sequence of actions is not evaluated. In general, however, within all this more recent research, the application of these techniques to sensor-based scenarios is far less studied.

Finally, some authors have combined temporal and non-temporal models in different ways. Mantyla et al., (2000) combined HMMs with self-organising maps (a particular kind of neural network) in order to be able to recognise static and dynamic gestures within one system, while Wu et al., (2016) used neural networks as state probability functions of an HMM to boost the expressive power of the model. These kinds of hybrid approaches can improve the classification performance, but ultimately they do not bring additional real-time capabilities to the HMM framework.

As will be detailed in chapter 5, our framework uses neural networks as the basis of its gesture recognition module. The relatively limited data complexity (a small number of sensor features), along with the simplicity, flexibility and speed of neural networks make them a powerful tool that also allows for quick prototyping and fine-tuning of the system.

2.4 Artificial Reasoning

The behaviour of a virtual agent can be modelled in a number of different ways. There are many cases where a well-crafted set of case-based rules can be enough, but, for some scenarios, attaining a reasonable conduct may require more sophisticated strategies. There is a long history of research in the area of artificial reasoning. One could cite classic planning strategies like the STRIPS algorithm (Fikes and Nilsson, 1971) as an early example. STRIPS allowed an agent to come up with a plan of action towards a specific goal in a mostly static and predictable environment. Although it is not, in itself, capable of dealing with dynamic changes during the execution of a plan, it became very popular and was the basis of other methods too, like hierarchical task-network planning (Erol, Hendler and Nau, 1994) for hierarchies of goals.

Working with real-time, dynamic contexts frequently call for more advanced models of the "mind" of the agent. Arguably, the most commonly referred paradigm in this area is belief–desire–intention (BDI), proposed first by Bratman, (1987) and then Rao and Georgeff, (1991). This framework describes agents as a compound of some knowledge about the world, or belief (commonly expressed as logical propositions), one or more

goals about the state of the world, or desire, and a set of plans. Each plan defines the conditions under which it may be executed and its expected results, and is composed of agent actions and possibly other intermediate goals. At any instant, an agent would update its beliefs base with information perceived from the environment, and then consider which of the feasible plans better satisfies its desires. Once a plan has been selected, it becomes the current intention of the agent until the next reconsideration takes place, when the intention might change in the light of new updated information. The important aspect here is that this model can accommodate for dynamic changes in beliefs and plan failures. This model inspired the Procedural Reasoning System (Georgeff and Ingrand, 1989), a particular implementation of BDI used in multiple real-world applications, as well as the belief–obligation–desire–intention framework (BOID) (Broersen et al., 2001), which introduces the concept of obligations to model externally imposed goals that may conflict with the agent's own.

There exists an ever more comprehensive family of approaches dubbed as "cognition architectures", which have been under research for a number of decades as well (Kotseruba and Tsotsos, 2016). These attempt to provide principled models that reproduce either the entirety or a significant part of the human intellect, spanning aspects like perception, learning, reasoning or memory. They are aimed to serve as a basis for more general forms of artificial intelligence, and some times are more of an exploratory effort into the foundations of a possible artificial mind than goal-oriented models addressing any particular situation. Although some of these models, such as SOAR (Laird, Newell and Rosenbloom, 1987), have found perdurable success in specific contexts, they are generally far too complicated for agents attempting to solve well-defined tasks in relatively simple environments.

For the purposes of modelling human–character interaction in VR, our framework includes an agent behaviour model based on state machines, described in chapter 5. Although our design would allow to integrate a more advanced reasoning model like BDI without affecting any of the other elements, state machines are already a popular model among game designers and animators, so ease of use was primed over expressiveness in this sense.

2.5 Neural Networks

Before moving to more concrete mathematical definitions, it is worth including here a brief introduction to neural networks as a general tool for the unfamiliar reader. Of all the techniques and algorithms in the field of machine learning, neural networks have

been by far the most popular and successful of the last decade. This accomplishment is all the more remarkable considering that, rather than a groundbreaking scientific innovation, the foundations of neural networks had already been laid out decades before their current hegemony. However, an accumulation of important advances in the theory, together with the relentless progress of hardware technology, have put neural networks in a position where they can be used to address nearly any kind of machine learning needs.

Neural networks were first introduced as a computational model for human neurons (McCulloch and Pitts, 1943). In particular, the design of the perceptron, the basic building block of neural networks, directly reflects the concepts of dendrites, axon and synapse. We now know that there is little real resemblance between neurons and perceptrons, but the mathematical model did constitute a general regression method for correspondences in real vector spaces. Neural networks are typically conceptualised as a collection of connected "units" arranged in "layers" but, in essence, a neural network is a series of consecutively applied linear mappings (the layers) which are followed by a differentiable non-linear function. The whole model can be expressed as an analytical function, where the coefficients of the linear mappings, initially random, are trainable parameters. Given a particular example of a regression problem (an input vector and its associated output vector), one can apply the neural network function to the input and compare the result to the expected output. This comparison can be given by a differentiable measure, such as the aggregated squared differences between both vectors. Using the gradient of this measure of the error with respect to the parameters of the model it is simple to define a gradient-based optimisation process to minimise it. The backpropagation algorithm (Werbos, 1974), which efficiently computes this gradient one layer at a time, made it possible to train relatively large models in practical scenarios using stochastic gradient descent.

The power of neural networks is not in their mathematical formulation as such, but in their ability to scale to the complexity of the problem. Indeed, it has been proven that, theoretically, any function can be approximated (to any given precision degree) by a sufficiently large network (Leshno et al., 1993). On paper, neural networks are a universal solution to regression. Yet the fact is that using them for any sizeable problem is frequently far from straightforward. In actuality, there is a myriad of practical concerns to navigate when building a neural network, including the size and structure of the network, the chosen activation function, the optimisation objective or the length and structure of the training. And while, mathematically, the learning potential of neural networks is independent of the particular encoding of the input and output, it

has been shown that different forms of preprocessing, such as normalisation, whitening or smart feature encoding can have a great impact on the results (Crone, Lessmann and Stahlbock, 2006), as do more intrinsic details like the parameter initialisation scheme (Fernández-Redondo and Hernández-Espinosa, 2000). And, in addition to that, neural networks also have to deal with more general issues common to any regression technique, such as overfitting or data imbalance.

These may be some of the reasons that hindered a widespread success of neural networks for a long time after their invention. However, the advances in the latest ten to twenty years have drawn a great deal of attention to them, to the point that it has become one of the most researched topics in machine learning, or even computer science in general. The field has had a resurgence in what is now called "deep learning", due to the architectural differences with respect to the original conception, and, leaving aside concerns that have been raised about potential abuses of this branch of research (Bengio, 2020), the truth is that a series of scientific and technological factors have made it possible for it to tackle very difficult problems with an unprecedented level of success. Among these, one of the most influential developments was the introduction of the convolutional neural network architecture for computer vision (LeCun et al., 1999), which cleverly repurposed the connectivity of the network units to take advantage of the structure in the data. This would become a key aspect of the prominent AlexNet model for object recognition (Krizhevsky, Sutskever and Hinton, 2012), a huge leap forward in the area that some recognise as a pivotal point for the "deep learning revolution". Similarly important have been the advances in recurrent neural networks (Rumelhart, Hinton and R.J. Williams, 1986), well suited to the analysis of sequential data as they feed back their output as an input for the next evaluation, and which play a crucial role in modern natural language processing (De Mulder, Bethard and Moens, 2015). In the area of reinforcement learning, the introduction of a deep Q-learning algorithm capable of playing video games with human-level performance (Mnih et al., 2015) sparked much research into applications of neural networks to control tasks. And generative models like variational autoencoders (Kingma and Welling, 2014) or generative adversarial networks (Goodfellow et al., 2014) are at the state of the art of data synthesis. Overall, these and other innovations, along with the capabilities of modern hardware, have made neural networks an extremely versatile and powerful machine learning technique.

Neural networks are at the centre of our research in this work, and in fact we will introduce a new kind of network architecture that improves on the results of existing conventional approaches through collections of parameters specialised over different subspaces of the domain of the problem at hand. Chapter 3 will describe this model in

detail, including the rationale behind it and its mathematical formulation.

2.6 Phase-Functioned Neural Networks

This section describes in detail the formulation of phase-functioned neural networks (PFNN), which will serve as the foundation for our grid-functioned neural network model described in chapter 3. The PFNN architecture was initially presented as a model for periodic processes. In its original context, this process was bipedal locomotion animation. The model ran on every frame of a simulation, receiving the current pose and control parameters (influenced by the user through a game controller) and outputting the next pose in the animation. The key insight in the architecture is that the behaviour of the neural network changes with the "phase". The phase, taken here in its mathematical sense, is simply a particular point within a cycle of the process. A PFNN contains four separate sets of parameters for a fixed neural network architecture, and associates each of these with a particular point in the cycle, so depending on the current phase the actual parameters to the neural network change. That is, at the beginning of a step, the parameters of the neural network that generates the next pose will be different to those used in the middle or ending stages of the same step. This change in the network parameters is done in a continuous way, so the transition is not apparent in the result.

The advantage of this model is that it avoids the averaging effect, very common to neural networks, by which high-frequency details in the output are lost in favour of a reduced overall error. In this case, that would mean biasing the pose towards a central position of the feet, producing valid but incorrect animation for most of the locomotion cycle. Using multiple sets of parameters, the particular details of each part in the cycle are much better preserved without impacting the rest of the animation.

The rest of this section outlines the mathematical formulation of the PFNN model as originally proposed by Holden, Komura and Saito, generalised to an arbitrary regression problem.

2.6.1 Multilayer Perceptron

Let us start by establishing the basic terms of a simple neural network model, or multilayer perceptron (MLP); please refer to appendix A for a more extended discussion of the concepts reviewed here. In a typical regression scenario we are faced with the problem of estimating an unknown function $f\colon \mathbb{R}^{D_{In}} \to \mathbb{R}^{D_{Out}}$ from a set of examples

$X \subset f$.[1] A simple neural network regressor $MLP\colon \mathbb{R}^{D_{In}} \to \mathbb{R}^{D_{Out}}$ is defined by an input size $D_{In} = D_0$, H hidden layers with sizes $D_1, \ldots, D_H$ and an output layer with size $D_{Out} = D_{H+1}$. The network is parameterised by a set of weight matrices $W_1 \in \mathbb{R}^{D_1 \times D_0}, \ldots, W_{H+1} \in \mathbb{R}^{D_{H+1} \times D_H}$ and bias vectors $\boldsymbol{b}_1 \in \mathbb{R}^{D_1}, \ldots, \boldsymbol{b}_{H+1} \in \mathbb{R}^{D_{H+1}}$. The total number of parameters in the model is therefore $T = \sum_{i=1}^{H+1} D_i D_{i-1} + D_i$. The neural network function is then expressed as:

$$\begin{aligned} MLP(\boldsymbol{x}; \boldsymbol{\theta}) &= \boldsymbol{x}^{H+1} \\ \boldsymbol{x}^0 &= \boldsymbol{x} \\ \boldsymbol{x}^i &= A_i(W_i \boldsymbol{x}^{i-1} + \boldsymbol{b}_i) \quad 0 < i \leq H+1 \end{aligned} \tag{2.1}$$

Where $\boldsymbol{\theta} \in \mathbb{R}^T$ is a vector containing all the parameters in the model and each $A_i\colon \mathbb{R}^{D_i} \to \mathbb{R}^{D_i}$ is a differentiable nonlinear activation function.

The parameters to the neural network $\boldsymbol{\theta}$ can be iteratively optimised to minimise a differentiable loss function $L\colon \mathbb{R}^{D_{Out}} \times \mathbb{R}^{D_{Out}} \to \mathbb{R}$. This loss function is the difference between the value regressed by the neural network and the value we desired to be produced by the network based on the current input. Again, much has been said about the different existing options for the loss function, but we may here consider the commonly chosen squared ℓ^2-norm.

$$L(\boldsymbol{y}, \widehat{\boldsymbol{y}}) = \|\boldsymbol{y} - \widehat{\boldsymbol{y}}\|^2 \tag{2.2}$$

Using the set of examples X, it is now straightforward to use any gradient-based optimisation strategy to train the model in order to approximate f. Although several more sophisticated methods exist, the process can be illustrated with a simple stochastic gradient descent process.

$$\boldsymbol{\theta}_{i+1} = \boldsymbol{\theta}_i - \mu \frac{1}{|B_i|} \sum_{(\boldsymbol{x}, \boldsymbol{y}) \in B_i} \nabla_{\boldsymbol{\theta}_i} L(MLP(\boldsymbol{x}), \boldsymbol{y}) \quad \forall i > 0 \tag{2.3}$$

Where each $B_i \subset X$ is a randomly sampled batch of examples and each $\boldsymbol{\theta}_i$ is a successive

[1] For simplicity, we consider here the regression of a deterministic function, but the general model and derivations are equally applicable to the more general probabilistic statement of the problem. We also ignore, for the extent of this exposition, the effects of noise, as well as other concerns such as overfitting.

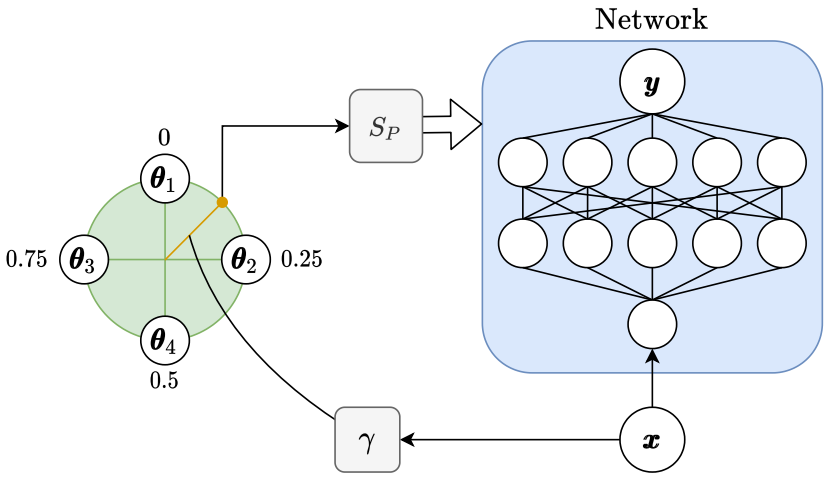


Figure 2-2: Diagram of a phase-functioned neural network. The γ function associates the input $\boldsymbol{x}$ to a point, or phase, in the experts cycle. The interpolated value of the expert parameterisations at that phase determines the effective parameterisation of the neural network.

approximation for the model parameters, $\boldsymbol{\theta}_1$ typically being a random sample from $\mathbb{R}^T$, e.g. $\boldsymbol{\theta}_1 \sim \mathcal{N}(\mathbf{0}, I_T)$. The value μ is the learning rate, expressing the scale of each optimisation step. Through consecutive increments to the parameters in the opposite direction to the gradient of the loss function, the neural network becomes a progressively better approximation of f.

2.6.2 Phase-Functioned Parameterisations

Now we consider a phase function $\gamma\colon \mathbb{R}^{D_{In}} \to [0,1]$ that characterises the regressed function f in some way. By that we mean that there is a strong (nonlinear) correlation between γ and f. That is, γ is some feature derived from the input that is known to have an important influence over the behaviour of f. For now, we assume that γ is a "periodic" property, in the sense that its value "cycles" through the interval $[0,1]$. More formally, we can simply say that there exists a continuous differentiable function $\gamma'\colon \mathbb{R}^{D_{In}} \to \mathbb{R}$ such that $\gamma(\boldsymbol{x}) = \gamma'(\boldsymbol{x}) - \lfloor \gamma'(\boldsymbol{x}) \rfloor$.

The exact form of γ is not important (as long as it is computable), but it should define a space where local approximation of f is possible. This means that a good local approximator of f for $\{\boldsymbol{x} \mid \gamma(\boldsymbol{x}) = a\}$ will generally perform well in a neighbourhood $\{\boldsymbol{x} \mid |\gamma(\boldsymbol{x}) - a| < \epsilon\}$. Knowing this, we can build an ensemble model composed of several parts specialised in different regions in the domain of γ, as represented by fig. 2-2. We define the set $\Theta = \{\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \boldsymbol{\theta}_3, \boldsymbol{\theta}_4\} \in \mathbb{R}^{T \times 4}$ containing four different sets of

parameters for the neural network, correspondingly associated with four evenly-spaced values, $\{c(1) = 0, c(2) = 1/4, c(3) = 1/2, c(4) = 3/4\}$, such that $\boldsymbol{\theta}_i$ is the expert set of parameters for the case where $\gamma(\boldsymbol{x}) = c(i)$. The question now is how to use and combine these parameterisations so they are appropriately trained and activated according to γ. One simple option would be to simply pick the parameterisation associated with the value closest (in a periodic sense) to the value of $\gamma(\boldsymbol{x})$. So, for $\gamma(\boldsymbol{x}) = 0.6$ we would pick $\boldsymbol{\theta}_3$, whereas for $\gamma(\boldsymbol{x}) = 0.9$ the model would use $\boldsymbol{\theta}_1$. This is a simple strategy, but it would result in a discontinuous model, which may be undesirable in many cases. Also, if we assume the modelled phenomenon is continuous, we may expect that, for a given value of γ, combining the surrounding parameterisations may yield a better estimation than using only one of them. We can do this in a C^1-continuous fashion with a cubic spline interpolation (Catmull and Rom, 1974). To do this, first we define the cubic spline function $S\colon [0,1] \times \mathbb{R}^{T\times 4} \to \mathbb{R}^T$.

$$\begin{aligned} S(\alpha, \boldsymbol{\tau}_1, \boldsymbol{\tau}_2, \boldsymbol{\tau}_3, \boldsymbol{\tau}_4) = \boldsymbol{\tau}_1 \left(-\frac{1}{2}\alpha + \alpha^2 - \frac{1}{2}\alpha^3\right) + \boldsymbol{\tau}_2 \left(1 - \frac{5}{2}\alpha^2 + \frac{3}{2}\alpha^3\right) \\ + \boldsymbol{\tau}_3 \left(\frac{1}{2}\alpha + 2\alpha^2 - \frac{3}{2}\alpha^3\right) + \boldsymbol{\tau}_4 \left(-\frac{1}{2}\alpha^2 + \frac{1}{2}\alpha^3\right) \end{aligned} \tag{2.4}$$

This function defines a T-dimensional curve from $\boldsymbol{\tau}_2$ (for $\alpha = 0$) to $\boldsymbol{\tau}_3$ (for $\alpha = 1$). Using S, we can build a chain of spline segments that form a C^1-continuous periodic interpolation of the four parameterisations in Θ as follows:

$$\begin{aligned} S_P(\gamma, \Theta) &= S(\alpha, \boldsymbol{\theta}_{t_1}, \boldsymbol{\theta}_{t_2}, \boldsymbol{\theta}_{t_3}, \boldsymbol{\theta}_{t_4}) \\ \alpha &= \beta - \lfloor \beta \rfloor \\ t_i &= 1 + (\lfloor \beta \rfloor + i - 2 \bmod 4) \\ \beta &= 4\gamma \end{aligned} \tag{2.5}$$

To break down the expression, γ falls in the interval between $c(t_2)$ and $c(t_3)$ (if we consider that $c(0)$ is at both ends of the domain of γ). $\boldsymbol{\theta}_{t_2}$ and $\boldsymbol{\theta}_{t_3}$ are therefore the closest experts to the left and right of γ respectively, while $\boldsymbol{\theta}_{t_1}$ and $\boldsymbol{\theta}_{t_4}$ are the second closest experts to the left and right respectively. The value $\alpha \in [0,1]$ is the relative position of γ within the interval. The definition of the spline interpolation function ensures that $S_P(c(i), \Theta) = \boldsymbol{\theta}_i$, all while preserving smooth continuity.

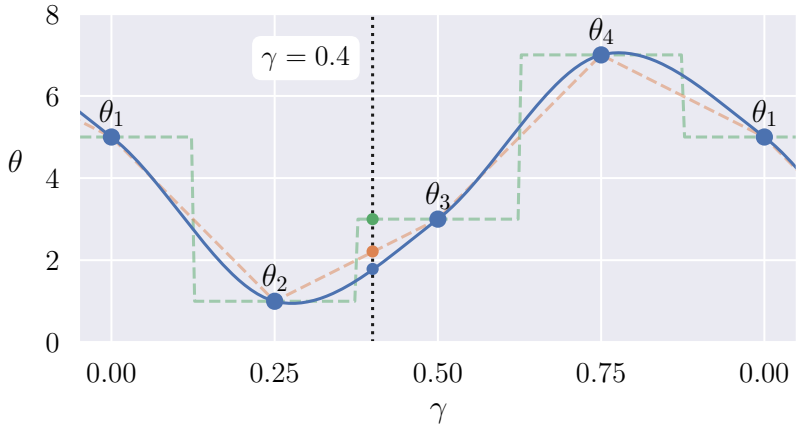


Figure 2-3: Comparison between spline parameter interpolation (blue) and linear (orange) and constant (green) piecewise parameter interpolation. Note the value of γ "cycles" through the interval $[0, 1]$. The black vertical line shows interpolated values for $\gamma = 0.4$.

Figure 2-3 shows a comparison between the spline formula above and simple constant or linear piecewise interpolation, with a set of four scalar parameters ($T = 1$). Even though a linear interpolation would also be continuous, the discontinuity in the gradient would be likely to produce unstable behaviour in the model. The spline assigns a smoothly varying weight to each expert parameterisation depending on γ, as shown in fig. 2-4, which results in a smooth overall result behaviour.

The interpolation of the four sets of parameters provides a new vector of parameters determined by γ, which we can in turn use to evaluate our model, simply by plugging it into the final model function $PFNN : \mathbb{R}^{D_{In}} \rightarrow \mathbb{R}^{D_{Out}}$:

$$PFNN(\boldsymbol{x}; \gamma, \Theta) = MLP(\boldsymbol{x}; S_P(\gamma(\boldsymbol{x}), \Theta)) \tag{2.6}$$

Note that this is still a differentiable function, and so the same training procedures apply to it. The different parameterisations in Θ become specialised simply because they are given more weight for particular values of γ, and so their gradients for these, which determine the scale of the parameter updates, will become larger. In other words, γ works as a gate for the impact of each parameterisation on the output, and therefore for the scale of their optimisation at every step.

Chapter 3 expands on the inherent limitations of PFNN and build on its definition to generalise the applicability of the model while maintaining its desirable locality features.

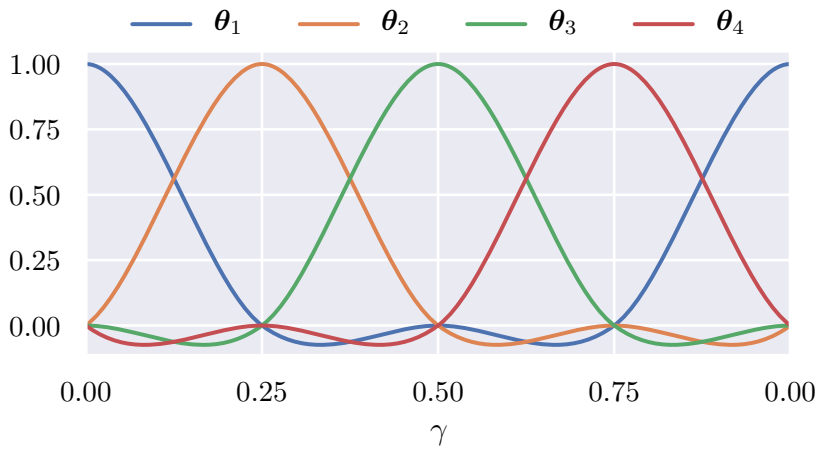


Figure 2-4: Weight assigned to each expert parameterisation across the phase domain.

2.7 Discussion

The works reviewed above are only a fraction of all the literature that may contribute towards the development of an interactive character. As stated initially, this is a very broad topic that could also encompass aspects like realistic human rendering, voice synthesis or muscle-based animation, among others. These are areas that are out of the scope outlined in chapter 1, but they come to show the extension of the problem in its more general form.

With respect to our approach to character animation and interaction, neural networks are an essential component of it. Theory and technology are at a point where they have become one of the most powerful and accessible machine learning methods to use, being a perfect fit for the data-driven aspects of the system. This contrasts with the more behaviourally grounded approach taken by other virtual human recreations, but it is better suited to our goals. There are numerous successful cases of neural networks applied to gesture recognition, and the new network architectures that have emerged for character animation showcase the potential of this approach. Even if the reasoning component is limited to state machines, the data-driven components are capable of handling all the additional complexity of the problem. We therefore have all the necessary pieces to begin putting together our interactive system in the following chapters.

Chapter 3

Grid-Functioned Neural Networks

One powerful idea in modern deep learning research is that of adding structure to the problem, both to the input data and the network architecture. Convolutional neural networks have their parameters arranged in "filters" that perform position-invariant pattern detection, which takes advantage of the inherent structure of the input data instead of blindly connecting every input value to every neuron. Recurrent neural networks saw a boost in their limited capability to track long-term dependencies thanks the long short-term memory model (Hochreiter and Schmidhuber, 1997), which introduced additional meaningful structure to the feedback loop. Variational autoencoders (Kingma and Welling, 2014) were able to improve over previous generative methods by endowing its latent vector with a probabilistic structure. This has also been true in the field of real-time 3D animation, and particularly in the area of locomotion for skeletal models, where several recent developments point in that direction. The idea behind the phase-functioned neural network (PFNN) architecture presented by Holden, Komura and Saito, (2017) can be seen as adding specific structure to a model in order to learn bipedal locomotion control from motion capture data. It partitions its parameters into specialisations for different stages or "phases" of the locomotion cycle, and then defines a continuous blend between these as walking progresses. The result is a smooth animation that avoids much of the detail loss associated with other architectures, showing that an ensemble of specialisations can be more effective than a larger general model.

The success of PFNN motivated further innovations along the same lines. When confronted with the problem of quadruped locomotion, Zhang et al., (2018) realised that it was not possible to define a single locomotion cycle, since the motion pattern of the four legs varied for different gaits. Instead, they proposed a mode-adaptive neural net-

work (MANN) that implicitly models the various locomotion modes. It also relies on a collection of specialisations, but in this case their purpose is opaque, as their mixture is given by a second neural network that decides how to combine them at each instant. Starke, Zhang et al., (2019) used a similar model to animate a biped character capable of interacting with its environment. In a sense, this approach offers more flexibility than the PFNN, but it also blurs its structure, as the purpose of each specialisation is no longer clear.

It is possible, however, to generalise PFNN into a more general architecture applicable to a wider variety of problems without losing sight of the meaning of our structure. The essential limitation in PFNN is that model specialisation can only happen along a single phase value, which may not fit well to different scenarios. But if we can break down the problem into more than one single specialisation axis, we might be able build a model with an explicit structure that captures all the variety in the data. We would then be working across multiple orthogonal facets or "dimensions" of the problem, and, in fact, this idea can be extended to any number of dimensions, accommodating arbitrarily complex scenarios.

We name this model grid-functioned neural network (GFNN), and in the following sections we derive its mathematical formulation, building on the definitions introduced at the end of chapter 2. At the end of this chapter, the effectiveness of this architecture is evaluated over a set of small synthetic problems, analysing its accuracy and performance qualities. The next chapters will show how this architecture can be used to model animation synthesis in different scenarios.

3.1 Gated Parameterisation Models

While PFNN showed great success in modelling locomotion, the specialisation of its parameterisations is constrained to one specific aspect of the problem, expressed by the phase. We could however conceive problems for which this is not enough, as the variety of the data cannot be characterised by a single factor. This was the case of Zhang et al., (2018) and Starke, Zhang et al., (2019), who tackled the problems of animating quadruped locomotion and environmental interaction respectively. These problems do not exhibit any obvious "phase" around which a PFNN can be built, so they require a different approach.

To address this limitation, the authors dissociated the concept of specialised parameterisations from the concept of phase. In order to select which parameterisations are activated at each time, a "gating network" is used instead. This network outputs a

set of weights that are used to linearly combine the parameterisations. Below are the mathematical details of this model.

3.1.1 Neural Parameterisation Activation

Continuing from the PFNN formulation in chapter 2, we would now want to have specialised parameterisations that are not tied to a particular phase value. We have then a set of M parameterisations to our architecture, $\Theta = \{\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_M\}$, which are, initially, not associated with any particular aspect of the problem. The number M is arbitrarily defined, but it should generally correlate with the complexity and variety of the problem. The combination of these parameterisations is given by a second "gating" neural network. The input to this network is a feature vector derived from $\boldsymbol{x}$ computed by a function $h\colon \mathbb{R}^{D_{In}} \to \mathbb{R}^{D'_{In}}$ (which can be the identity function), and it outputs a vector in $[0,1]^M$. Since this output will be used to compute the combination of the parameterisations of the main network, its values should add up to one in order to preserve the overall scale. One simple way to achieve this is to use the softmax function as activation for the output layer:

$$\text{Softmax}(\boldsymbol{a})_i = \frac{e^{a_i}}{\sum_{j=1}^{M} e^{a_j}} \tag{3.1}$$

The gating network has its own architecture with H' layers with sizes $D'_1, \ldots, D'_{H'}$, and thus its own vector of parameters, $\boldsymbol{\theta}' \in \mathbb{R}^{T'}$, where T' is the total number of parameters for this architecture. The gating vector for the parameterisations is then $MLP(h(\boldsymbol{x}); \boldsymbol{\theta}')$, and the expression of the model function is given by:

$$MLP\left(\boldsymbol{x}; \sum_{i=1}^{M} \boldsymbol{\theta}_i \cdot MLP(h(\boldsymbol{x}); \boldsymbol{\theta}')_i\right) \tag{3.2}$$

Figure 3-1 shows a representation of the described formulation. As a differentiable model, all of its parameters can be trained as usual. However, as there is no phase directing the activation of the different parameterisations, their specialisations emerge from the training process itself. On the one hand, that is much more flexible, because it is possible for the network to put the focus on unrelated facets of the problem. However, we do not have an intuition about what these facets are, or even whether some of the parameterisations are redundant or unused.

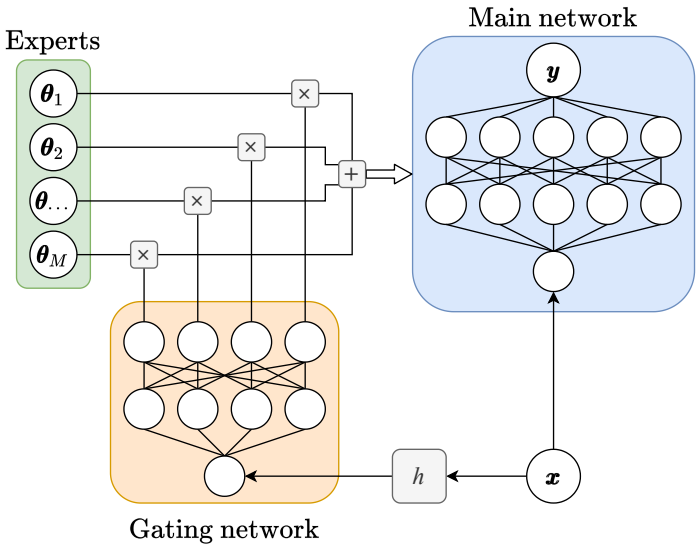


Figure 3-1: Diagram of a gated parameterisation model. The gating network, evaluated on a function h of the input, regulates a linear combination of the expert parameterisations. The resulting vector of parameters is then used by the main network to compute the final output.

3.2 Grid-Functioned Neural Networks

We would like to create a model that maintains the control over the division of the problem into parameterisations given by PFNN while being applicable to more diverse problems, like the gated parameterisation models. We have already seen that there are scenarios where a single phase value is not enough, but there are other limitations to the original PFNN formulation too. On the one hand, it fixes the number of parameterisations to four, but one could argue that some problems may be better modelled with more or fewer specialisations. On the other hand, it is focused on periodic processes, but it would be just as reasonable to consider the possibility of specialising in non-periodic features. For example, the animation of a character may be very strongly conditioned by a "stamina" factor, which is not a periodic value but should see a smooth behaviour progression between its minimum and maximum value.

The GFNN model generalises PFNN to work along multiple specialisation dimensions, with periodic and non-periodic features and an arbitrary number of divisions across each dimension. It therefore generalises the ability of PFNN to break down a problem into parts to arbitrarily complex domains. 2D and 3D spatial functions are good examples of problems where a multidimensional structure like GFNN is the most natural arrange-

ment for a set of experts. In general, such kind of problem where our prior knowledge of the data allows us to define a set of principal axes that articulate the behaviour of the function may benefit from the GFNN architecture. While gated parameterisation models attempt to "discover" these axes as part of the training, encoding them into the model structure guarantees that the learned function will respond reliably to changes in that space.

In addition to deriving its mathematical construction, this section characterises the time and space complexity of the model and some approximations to it.

3.2.1 Grid-Functioned Parameterisations

Following from our previous notation, we now consider the case where there are multiple phase-like attributes in our problem. In the simplest case, we could have just two of them, $[\gamma_1(\boldsymbol{x}), \gamma_2(\boldsymbol{x})]$. These are both functions with similar properties to the phase in a PFNN, with the difference that now each $\gamma_i \colon \mathbb{R}^{D_{In}} \to [0, 1]$ is not required to be periodic. Applying the PFNN architecture, we could have one ensemble of experts for γ_1 and another one for γ_2, but that does not immediately give us a method to combine the two of them to obtain a superior model. We could take the interpolated parameterisations for γ_1 and γ_2 and then average them in some way, but there is really no reason to believe this is a meaningful operation, as each one is specialised in a different, independent aspect of the problem. And, after all, if γ_1 and γ_2 both condition the behaviour of the regressed function, if follows that the experts in γ_1 should be conditioned by γ_2, and vice versa.

Indeed, what we find here is a two-dimensional space where, in the general case, γ_1 and γ_2 my take any pair of values in $[0, 1]^2$. So it makes sense that our experts are distributed across this space, and not across two independent one-dimensional spaces. This is represented in the diagram in fig. 3-2. We define a two-dimensional grid $G_2 = \{1, \ldots, N_1\} \times \{1, \ldots, N_2\}$, where N_1 and N_2 are the number of grid divisions across the domains of γ_1 and γ_2 respectively. Each element of the grid has an associated expert parameterisation, $\Theta_2 = \{\boldsymbol{\theta}^2_{\boldsymbol{g}} \in \mathbb{R}^T \mid \boldsymbol{g} \in G_2\}$. Each of these parameterisations is to be the expert for a particular pair of values of $\gamma_1(\boldsymbol{x})$ and $\gamma_2(\boldsymbol{x})$, given by the function $c \colon G_2 \to [0, 1]^2$ defined as:

$$c(\boldsymbol{g})_i = \begin{cases} (g_i - 1)/N_i & \text{if } \gamma_i \text{ is periodic} \\ (g_i - 1)/(N_i - 1) & \text{otherwise} \end{cases} \tag{3.3}$$

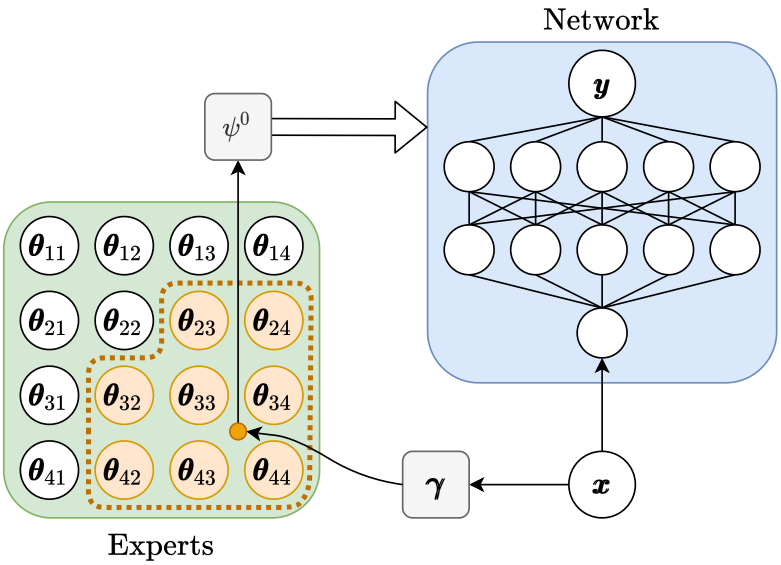


Figure 3-2: Diagram of a 4×4 grid-functioned neural network. Each input $\boldsymbol{x}$ is projected to a point in the expert grid space through the $\boldsymbol{\gamma}$ functions. The neighbouring expert parameterisations (in orange) are combined through multi-dimensional cubic interpolation into the final effective neural network parameterisation.

Note that, according to this expression, the domain of a periodic γ_i would be split in N_i intervals, $\{[c(1)_i = 0, c(2)_i), \ldots, [c(N_i - 1)_i, c(N_i)_i), [c(N_i)_i, 1)\}$, whereas a non-periodic γ_i would be split into $N_1 - 1$ intervals, $\{[c(1)_i = 0, c(2)_i), \ldots, [c(N_i - 1)_i, c(N_i)_i = 1]\}$.

The mixture of the parameterisations in Θ_2 is built analogously to the one-dimensional case, through two successive spline interpolations. First, we reformulate the interpolation expression from eq. (2.5) in a more general form:

$$
\begin{aligned}
S_G(\gamma_i, \{\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_{N_i}\}) &= S(\alpha, \boldsymbol{\theta}_{t_1}, \boldsymbol{\theta}_{t_2}, \boldsymbol{\theta}_{t_3}, \boldsymbol{\theta}_{t_4}) \\
\alpha &= \beta - \lfloor \beta \rfloor \\
t_i &= \begin{cases} 1 + (\lfloor \beta \rfloor + i - 2 \bmod N_i) & \text{if } \gamma_i \text{ is periodic} \\ \min(\max(\lfloor \beta \rfloor + i - 1, 1), N_i) & \text{otherwise} \end{cases} \\
\beta &= \begin{cases} N_i \gamma_i & \text{if } \gamma_i \text{ is periodic} \\ (N_i - 1)\gamma_i & \text{otherwise} \end{cases}
\end{aligned}
\tag{3.4}
$$

We can now apply this interpolation first to obtain the set of intermediate expert parameterisations $\Psi^1 = \{\boldsymbol{\psi}_g^1 \in \mathbb{R}^T \mid g \in \{1, \ldots, N_1\}\}$ as:

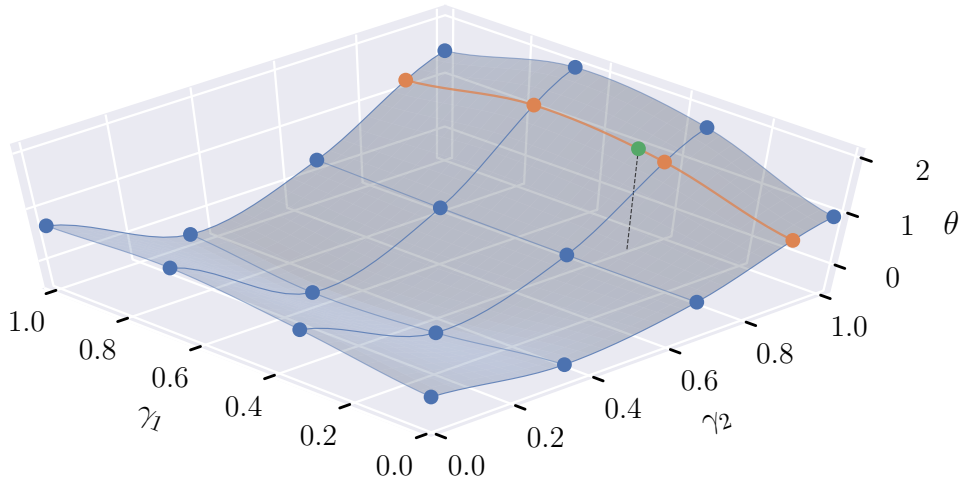


Figure 3-3: Two-dimensional spline parameter interpolation for $\gamma_1 = 0.4$ and $\gamma_2 = 0.9$. Blue dots represent different values of a scalar parameter θ across a 4×4 grid. The orange dots are the result of the first interpolation step, which reduces the problem back to one dimension, and the green dot is the final interpolated value.

$$\boldsymbol{\psi}_i^1 = S_G(\gamma_2, \{\boldsymbol{\theta}_{i1}^2, \ldots, \boldsymbol{\theta}_{iN_2}^2\}) \tag{3.5}$$

Ψ^1 represents a set of experts on γ_1 for the current value of γ_2.[1] This reduces the problem to a one-dimensional interpolation again, and so the fully interpolated parameterisation would simply be $S_G(\gamma_1, \Psi^1)$.

Figure 3-3 shows an example of this two-dimensional spline interpolation process for a 4×4 grid. In the first step the problem is reduced from the two dimensions γ_1 and γ_2 to only γ_1, by computing the grid points of a one-dimensional spline interpolated for the precise value of γ_2. From there, obtaining the actual interpolated value is straightforward.

Note how, like PFNN, the grid approach attempts to structure the problem space according to the distribution of the experts. That is the key feature that enables it to capture local patterns more effectively, but it can too increase the bias in the model. This may happen when the γ_i functions do not represent meaningful dimensions of the problem, so it is essential that their definition is well suited to the nature of the data.

Proceeding in the same manner as above, we can generalise this reasoning to an arbitrary

[1] It would also be possible to define a set of experts on γ_2 for the current value of γ_1 instead. Mathematically, the final result would be identical.

number of dimensions K, with a characterising vector $\boldsymbol{\gamma} = [\gamma_1(\boldsymbol{x}), \ldots, \gamma_K(\boldsymbol{x})] \in [0,1]^K$. We would now have a grid $G_K = \{1, \ldots, N_1\} \times \ldots \times \{1, \ldots, N_K\}$, N_i being the number of divisions across the domain of γ_i. The set of parameterisations $\Theta_K = \{\boldsymbol{\theta}^K_{\boldsymbol{g}} \in \mathbb{R}^T \mid \boldsymbol{g} \in G_K\}$ contains the $\prod_{j=1}^K N_j$ experts across the grid, each associated with a value $c(\boldsymbol{g}) \in [0,1]^K$ defined as in eq. (3.3). The interpolation of Θ_K for $\boldsymbol{\gamma}$ across the K dimensions of the grid is then built recursively as follows:

$$\begin{aligned} \psi^K_{\boldsymbol{g}}(\boldsymbol{\gamma}) &= \theta^K_{\boldsymbol{g}} \\ \psi^i_{j_1 \ldots j_i}(\boldsymbol{\gamma}) &= S_G(\gamma_{i+1}, \{\psi^{i+1}_{j_1 \ldots j_i 1}, \ldots, \boldsymbol{\psi}^{i+1}_{j_1 \ldots j_i N_{i+1}}\}) \quad 0 \leq i < K \end{aligned} \tag{3.6}$$

Each $\Psi_i(\boldsymbol{\gamma}) = \{\psi^i_{\boldsymbol{h}}(\boldsymbol{\gamma}) \mid \boldsymbol{h} \in \mathbb{N}^{N_1 \times \ldots \times N_i}\}$ is an i-dimensional grid of parameterisations obtained after $K - i$ dimensions have been interpolated. The process ends at $\psi^0(\boldsymbol{\gamma}) = S_G(\gamma_1, \Psi_1(\boldsymbol{\gamma}))$, and the final expression of the grid-functioned neural network model $GFNN : \mathbb{R}^{D_{In}} \to \mathbb{R}^{D_{Out}}$ is:

$$GFNN(\boldsymbol{x}; \Theta_K) = MLP(\boldsymbol{x}; \psi^0(\boldsymbol{\gamma})) \tag{3.7}$$

As before, this expression is differentiable with respect to every parameter in Θ_K, so the model can still be trained with the same methods as a regular multilayer perceptron. Through the multiple phase values, each training example is deterministically assigned to a particular mixture of parameterisations, so the specialisation of each of them is always well defined.

3.2.2 Computational Cost and Approximations

Using a grid-functioned model has an impact on the computational complexity of the model, both in terms of time and space. A base architecture requiring T parameters, applied over a grid with size $N_1 \times \ldots \times N_K$, would have a total of $T_K = T \prod_{j=1}^K N_j$ parameters. Generally, grid-functioned models should work on smaller architectures than regular models tackling the same problem, since each of the parameterisations would specialise in a smaller subdomain of the problem. This opens a design space for the model where, given a budget of parameters, we may choose how to distribute them across different dimensions broken down into different numbers of grid divisions.

We must consider, however, the computational cost that the cubic spline interpolation adds to the inference process. This is particularly important for our goals, since we are

interested in real-time applications. We can estimate this cost as the total number of required floating point operations. Looking back at eq. (3.6), computing one particular $\boldsymbol{\psi}^i_{j_1 \dots j_i}$ $(0 \leq i < K)$ requires interpolating N_{i+1} parameterisations. Reviewing eq. (3.4) and eq. (2.4), this interpolation requires the computation of four interpolation weights (the coefficients of each $\boldsymbol{\tau}$ term) for the four parameterisations located closest to the current value of γ_{i+1}. These four weights take the same value for every interpolation computed as part of Ψ^i, and they take a fixed number of floating point operations F_W to compute. The aggregation of the T parameters itself according to those weights in eq. (2.4) takes four multiplications and three additions per parameter, totalling to $7T$. The computation of each $\Psi_i(\boldsymbol{\gamma})$ requires interpolating $\prod_{j=1}^{i} \min(N_j, 4)$ grid parameterisations (for grid dimensions with more than four divisions only the four closest points across that dimension participate in the computation each time). Putting everything together, the computational cost of the cubic spline interpolation F_{Interp} results in:

$$F_{Interp} = \sum_{i=0}^{K-1} \left(F_W + 7T \prod_{j=1}^{i} \min(N_j, 4) \right) \tag{3.8}$$

Obviously, the term with the largest impact in the above formula is T, as the number of parameters of even modest architectures is rarely below the thousands, if not hundreds of thousands or more. The grid dimensions, being a multiplicative factor, is also an important factor in the time complexity. Note that in some cases it is possible to simplify some of the computation. A singleton grid dimension would not require any interpolation (nor break down the problem in any way) and, in the case of grid dimensions with two or three divisions, the formula in eq. (2.4) could be rewritten for faster computation (as some of the interpolated $\boldsymbol{\tau}$ values would be the same). However, F_{Interp} represents the cost for the general case.

It is worth pointing out that, for grids with less than four divisions in some dimension, the order in which dimensions are interpolated also affects the cost. From the point of view of floating point operations, having five or more divisions across some grid dimension incurs the same cost as having four. But with grid dimensions with two or three divisions, the order of the dimensions may change the value of the product at the end of each summed term in eq. (3.8). For example, consider a two-dimensional grid with size 4×2. In the first interpolation step $(i = 1)$, the grid is reduced to one dimension with four divisions, computed through four corresponding interpolations. In the next step $(i = 0)$, the grid is reduced to a single point computed through another interpolation. Thus, five interpolations are required, and so we have $F_{Interp} = 2F_W + (7T)5$. However,

for a grid with shape 2×4 only two interpolations are required in the first step, and the cost would boil down to $F_{Interp} = 2F_W + (7T)3$. For this reason, it is recommendable, in the case of heterogeneous grid sizes, to reorder the dimensions of the grid in ascending order for the purposes of interpolation, reducing first the largest dimensions.

In any case, the cost of the spline interpolation may prove too high in some circumstances. In those cases, we may be willing to sacrifice some computation accuracy for speed. We can replace the spline interpolation operation with simpler alternatives, namely linear and constant piecewise interpolations. However, applying these over the existing grid of parameterisations would yield an overly crude approximation. Instead, we use an augmented, denser grid, computed from the original one. Given our grid G_K and parameterisations Θ_K, we define the new grid $G'_K = \{1, \ldots, N'_1\} \times \ldots \times \{1, \ldots, N'_K\}$, with $N'_i \geq N_i$. Each element of the augmented grid becomes then the expert associated to a value $c' \colon G'_K \to [0,1]^K$ defined as usual:

$$c'(\boldsymbol{g})_i = \begin{cases} (g_i - 1)/N'_i & \text{if } \gamma_i \text{ is periodic} \\ (g_i - 1)/(N'_i - 1) & \text{otherwise} \end{cases} \tag{3.9}$$

The parameterisations for this grid, $\Theta'_K = \{\boldsymbol{\theta}'^K_{\boldsymbol{g}} \mid \boldsymbol{g} \in G'_K\}$ are simply computed from the original ones as before:

$$\boldsymbol{\theta}'^K_{\boldsymbol{g}} = \psi^0(c'(\boldsymbol{g})) \tag{3.10}$$

Using the denser grid of parameterisations Θ'_K, it is not difficult to compute the linear interpolation function $\psi'_{Linear} \colon [0,1]^K \to \mathbb{R}^T$ as follows:

$$\begin{aligned} \psi'_{Linear}(\boldsymbol{\gamma}) &= \sum_{\boldsymbol{q} \in \{1,2\}^K} \theta'_{\boldsymbol{t}(\boldsymbol{q})} \prod_{i=1}^{K} \alpha(\boldsymbol{q})_i \\ \alpha(\boldsymbol{q})_i &= \begin{cases} 1 - (\beta_i - \lfloor \beta \rfloor) & \text{if } q_i = 1 \\ \beta_i - \lfloor \beta \rfloor & \text{if } q_i = 2 \end{cases} \\ t(\boldsymbol{q})_i &= \lfloor \beta_i \rfloor + q_i \bmod N'_i \\ \beta_i &= \begin{cases} N'_i \gamma_i & \text{if } \gamma_i \text{ is periodic} \\ (N'_i - 1)\gamma_i & \text{otherwise} \end{cases} \end{aligned} \tag{3.11}$$

Here, the 2^K parameterisations in the smallest hypercube around $\boldsymbol{\gamma}$ in the grid are selected with $\boldsymbol{t}(\boldsymbol{q})$. Each of them is scaled by a weight which is the product of the corresponding linear weights across all dimensions, and then all are aggregated into the full interpolation. The computation of each linear weight $\alpha(\boldsymbol{q})_i$ takes a fixed number of operations F_L, and their product are $K-1$ multiplications. In sum, the total number of floating point operations required by the piecewise linear interpolation F_{Linear} is:

$$F_{Linear} = 2^K \left(T + KF_L + K - 1\right) \tag{3.12}$$

The case of piecewise constant interpolation (also called nearest-neighbour interpolation) is even simpler, as it just involves picking the closest parameterisation in the grid. The interpolation function $\psi'_{Constant} \colon [0,1]^K \to \mathbb{R}^T$ is just:

$$\begin{aligned} \psi'_{Constant}(\boldsymbol{\gamma}) &= \boldsymbol{\theta'_t} \\ t_i &= \lfloor \beta_i \rceil + 1 \bmod N'_i \\ \beta_i &= \begin{cases} N'_i \gamma_i & \text{if } \gamma_i \text{ is periodic} \\ (N'_i - 1)\gamma_i & \text{otherwise} \end{cases} \end{aligned} \tag{3.13}$$

In this case, the only necessary computation is the closest position in the grid. The number of floating point operations $F_{Constant}$ is then proportional to the number of dimensions in the grid by a constant factor F_C:

$$F_{Constant} = KF_C \tag{3.14}$$

Figure 3-4 shows these approximations applied to the parameter interpolation from fig. 2-3. Using a denser grid, the linear and constant piecewise approximations can result in a fairly accurate interpolation for a fraction of the computational power.

3.3 Evaluation

To analyse the capabilities of GFNN as a general machine learning model, in particular as compared to a traditional MLP architecture, we study its application to a selection of small and well-known problems.[2] If we train two neural networks with a similar number

[2]Code and data available at `https://doi.org/10.15125/BATH-00752` (Dehesa, 2020).

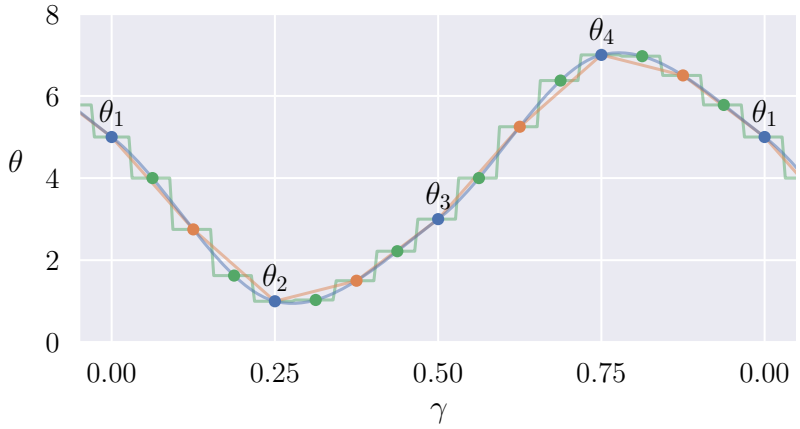


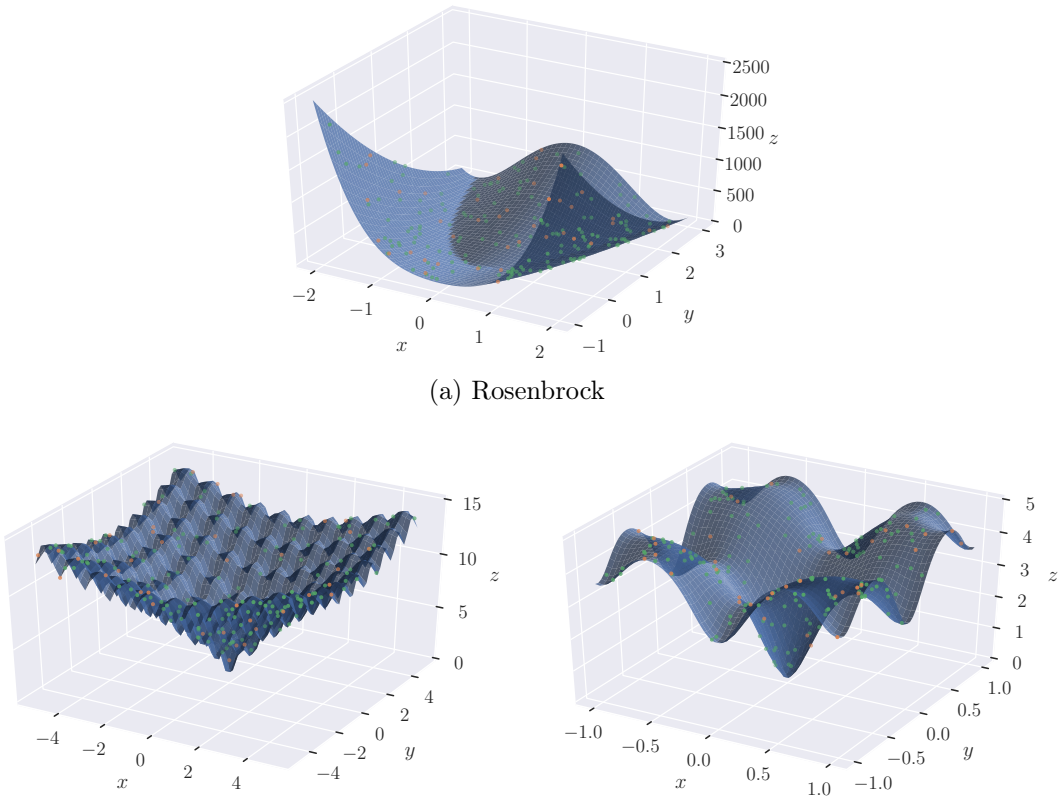
Figure 3-4: One-dimensional parameter interpolation using spline interpolation (blue) and linear (orange) and constant (green) piecewise approximations. Original grid has four points, linear approximation uses eight points and constant approximation uses sixteen points.

of parameters on the same data, and fix the activation functions, training procedure and other details, we can say that the architecture of the network yielding the better performance is superior, at least for the particular problem at hand.

The Rosenbrock function (Rosenbrock, 1960) and the Ackley function (Ackley, 1987) are real-valued 2D functions typically used as optimisation performance tests, but they can also serve the purpose of benchmarking the GFNN architecture. They are good choices to test our model as they are, as shown below, difficult to learn for traditional neural networks, but their low dimensionality makes it possible to visualise the learned models and analyse their behaviour better than the numerical error statistics alone would allow (the following chapters will showcase the capabilities of GFNN in more complex scenarios). The functions are defined by the following expressions:

$$\begin{aligned} \mathit{Rosenbrock}(x, y) &= (1 - x^2) + 100(y - x^2)^2 \\ \mathit{Ackley}(x, y) &= -20 \exp\left(-\sqrt{\frac{x^2 + y^2}{50}}\right) - \exp\left(\frac{\cos 2\pi x + \cos 2\pi y}{2}\right) + e + 20 \end{aligned} \tag{3.15}$$

Figure 3-5 shows three small regression problems based on these functions. For each of the shown functions, a random selection of 200 points from the displayed region is uniformly sampled, split 80% for training and 20% for evaluation. The goal of the



(a) Rosenbrock

(b) Ackley

(c) Ackley (small region)

Figure 3-5: Evaluation functions. Green points are training data, orange dots are evaluation data.

problem is to reconstruct each of the functions within the given region. The Rosenbrock function (fig. 3-5a) is fairly smooth, but becomes very steep towards the negative y axis. The Ackley function (fig. 3-5b) has an overall simple conic shape, but its surface is distorted by a high-frequency sinusoidal pattern. Taken in a smaller region (fig. 3-5c), it appears as a less noisy but highly nonlinear surface.

In order to evaluate the effectiveness of using a grid model for this problem, we can train different models with different grid sizes but maintaining an approximately fixed number of parameters. That is, the only difference between the models is the way in which the parameters are used. Given the simplicity of the problems, we can just use all input features to function a 2D grid in the models (which in this case would work as non-periodic features). Table 3-1 lists different configurations for 2D grid models with two inputs units, one output unit and two hidden layers. All configurations have a total of approximately ten thousand parameters, and require a variable number of floating point operations depending on the structure. Note that a grid size of 1×1 (or any

Grid size	Hidden layers	Parameters	Flops
1×1 (MLP)	100, 100	10 501	20 801
2×2	48, 48	10 180	27 960
3×3	31, 31	10 053	24 587
4×4	23, 23	10 320	23 852
5×5	18, 18	10 375	15 352
6×6	14, 14	9 612	9 884
7×7	12, 12	10 045	7 594

Table 3-1: Evaluated models. All models have two input units and one output unit.

grid size with singleton dimensions only) is equivalent to a regular MLP with no grid interpolation, and so this model can be used as a baseline to measure the results of the rest of actual GFNN models.

Each of these configurations is evaluated against each of the problems in exactly the same way. All models use ReLU activation for their hidden layers and linear activation in the output. Input and output data is normalised with respect to the mean and standard deviation of the training data. GFNN models expect grid function values to be in the unit interval, so the input features cannot be directly used for this purpose, even after normalisation. Instead, a "squashed" version of the input features is used, applying the sigmoid function to their normalised values. This results in a pair of values that appropriately map the input space to the unit interval.

Every model is trained on each problem to minimise the mean squared error at the output using Adam optimisation (see appendix A). The training runs for 100 000 steps, with a batch size of 32 examples per step and a fixed learning rate of 0.001. For the sake of simplicity, and to reduce the amount of confounding factors affecting the results, no regularisation mechanisms are used in this case. These are also less relevant in this context than in real situations in any case, given the low dimensionality of the problems and the lack of noise in the synthetic data.

Table 3-2 shows the results of the experiment, and fig. 3-6 provides a visual representation of the distribution of the error in each case. In general, all GFNN models significantly outperform the MLP model, with a trend favouring grids of medium density. There are differences between the problems. For Rosenbrock, a grid size of 5×5 appears to be optimal. For the first Ackley problem, both 4×4 and 7×7 yield similar statistics, although we will see they exhibit a rather different behaviour. In the case of the small region variant of the Ackley problem, the 5×5 and 7×7 grids offer the best results. In this problem, it is interesting to note how odd grid sizes appear to

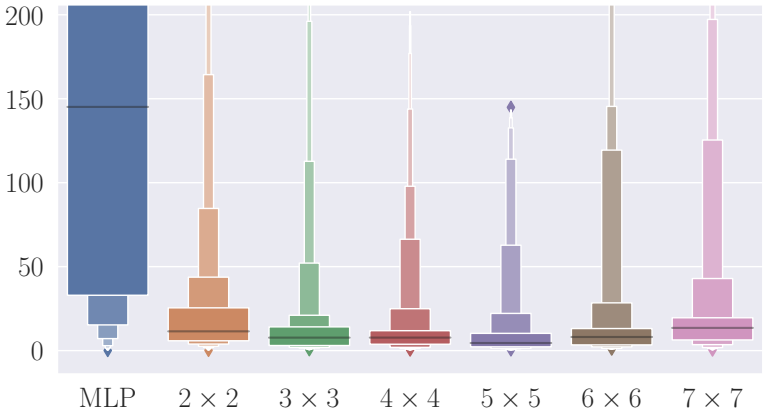
Model	Rosenbrock			Ackley			Ackley (small region)		
	Median	Mean	SD	Median	Mean	SD	Median	Mean	SD
MLP	145.1	266.5	368.3	1.723	1.776	1.083	0.393	0.424	0.228
2×2	11.5	32.4	68.6	0.491	0.586	0.418	0.073	0.125	0.187
3×3	7.7	20.6	50.6	0.479	0.581	0.402	0.028	0.068	0.151
4×4	7.8	18.2	36.7	0.411	0.582	0.428	0.030	0.079	0.164
5×5	4.5	15.9	30.9	0.492	0.689	0.573	0.018	0.052	0.110
6×6	8.1	26.0	59.7	0.557	0.734	0.625	0.035	0.079	0.112
7×7	13.5	43.6	120.0	0.411	0.615	0.492	0.020	0.057	0.106

Table 3-2: Error statistics of the evaluated models for each problem.

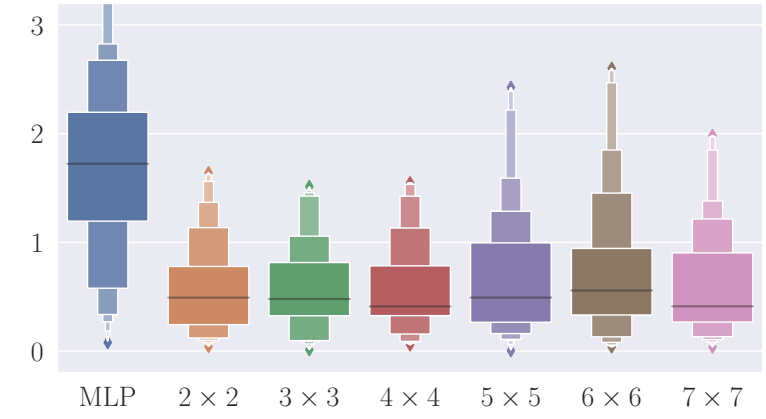
perform better than even sizes. This is actually coherent with the shape of the function itself, with a global minimum at the centre of the considered regions. In an odd-sized grid, there will be a grid point associated with the exact centre of the function domain, making this pattern easier to learn. This highlights the correspondence between the grid dimensions and the domain of the problem. Note also that both the 5×5 and 7×7 models take fewer floating point operations than the MLP model to compute, adding a reduction in the computational complexity to the improved regression accuracy.

The numerical metrics can also be visually appreciated in figures figs. 3-7 to 3-12 (pages 41 to 46), which show the complete reconstructions and error heat maps for each grid size and problem. The reconstructions show the value computed by each of the models over the whole domain of the problem, while the heat maps indicate the magnitude of the error at each point in a chromatic scale. In general, it can be seen how the MLP model is simply not capable of matching the function shape, but the densest grids appear closer to what we would consider an overfitted model. This is particularly evident in the case of the first Ackley problem (fig. 3-9). Here, neither the MLP nor GFNN models are capable of reproducing the detailed texture of the function surface, but 3×3 and 4×4 grid models produce a fair approximation to the overall shape. The 7×7 grid, on the other hand, which did achieve similarly good numerical results, shows a principle of the wavy pattern, but still far from the actual function, and at the expense of significant deviations in some of the outer regions. In the small region problem this does not become an issue, though. In this case GFNN models are able to match the function shape very closely over the whole domain, as local patterns remain smooth in all cases, and the error is generally very low, in dire contrast with the poor reconstruction by the MLP model.

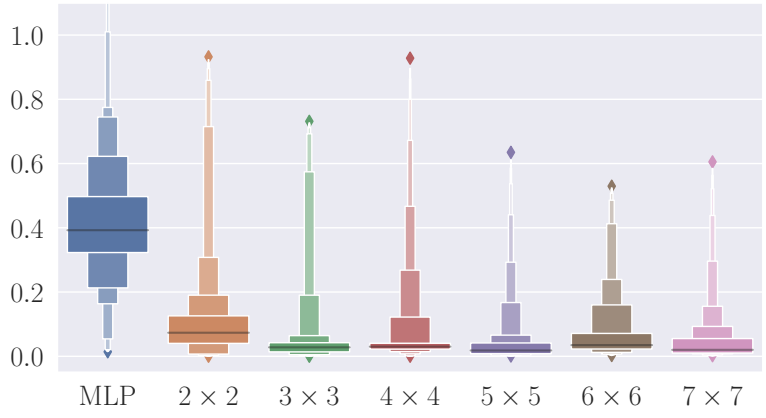
Going back again to the first Ackley problem, it may be interesting to analyse whether



(a) Rosenbrock

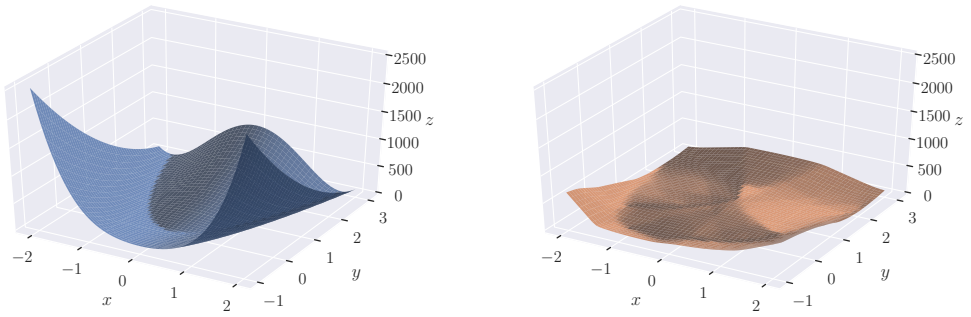


(b) Ackley



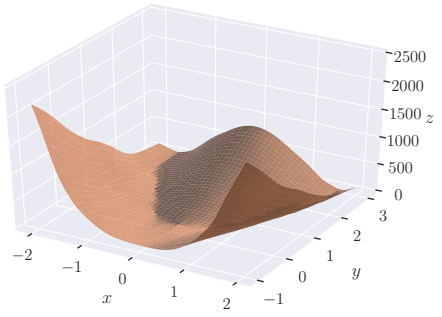
(c) Ackley (small region)

Figure 3-6: Error distribution of each model for each of the problems.

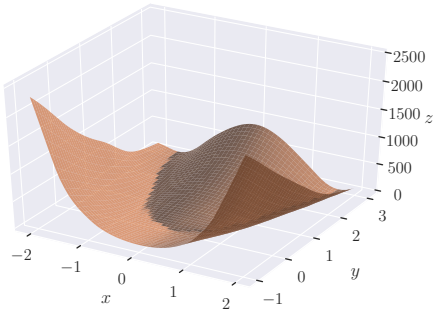


(a) Function

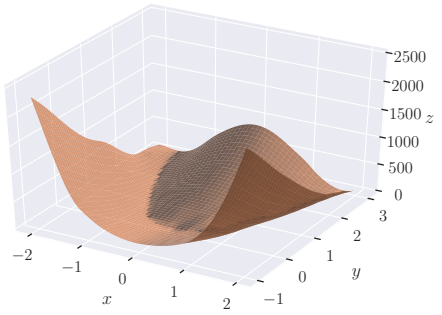
(b) MLP



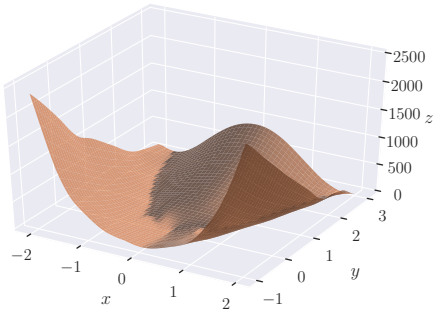
(c) 2×2



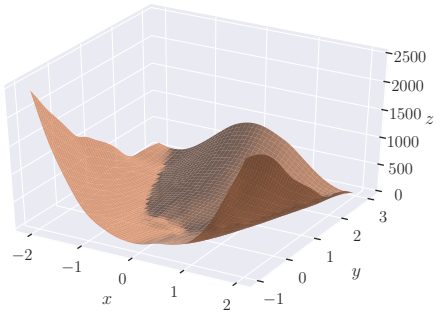
(d) 3×3



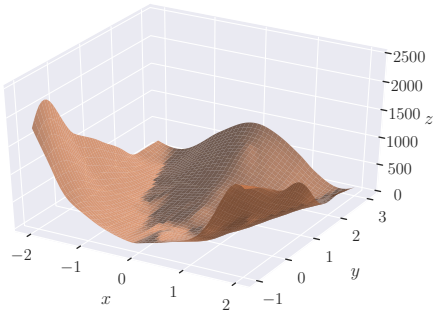
(e) 4×4



(f) 5×5



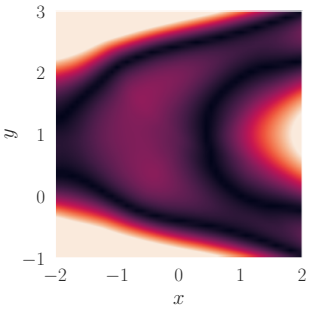
(g) 6×6



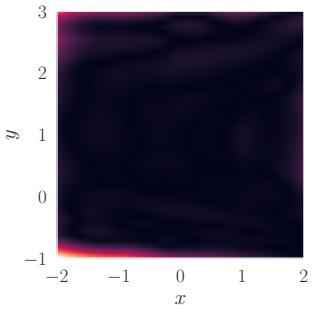
(h) 7×7

Figure 3-7: Reconstruction of the Rosenbrock function by an MLP and different GFNN models. Figure (a) shows the original function.

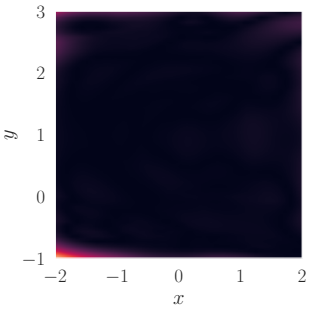
(a) Error scale



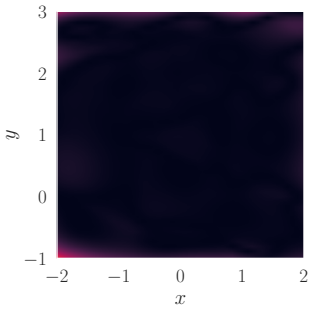
(b) MLP



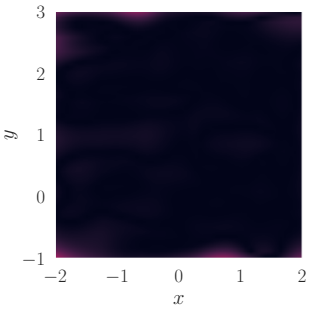
(c) 2×2



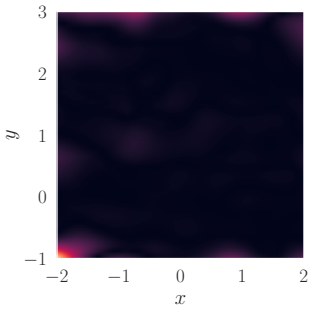
(d) 3×3



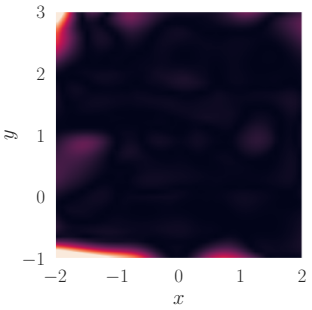
(e) 4×4



(f) 5×5

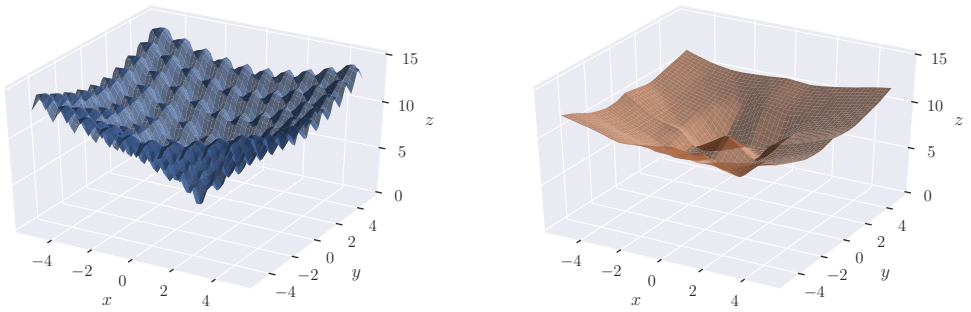


(g) 6×6



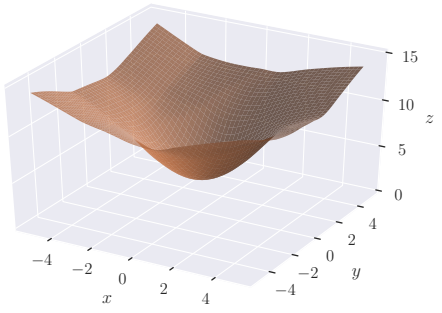
(h) 7×7

Figure 3-8: Error map for the reconstruction of the Rosenbrock function by an MLP and different GFNN models.

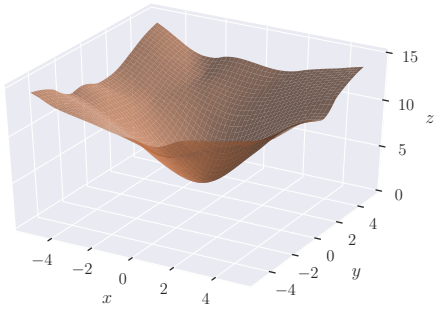


(a) Function

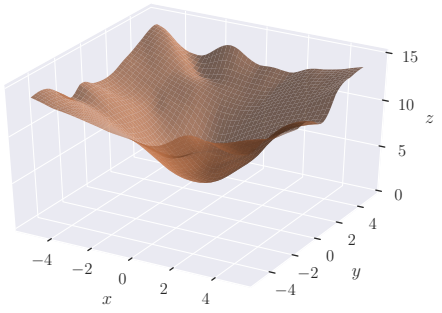
(b) MLP



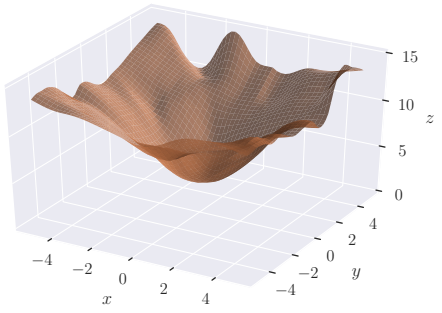
(c) 2×2



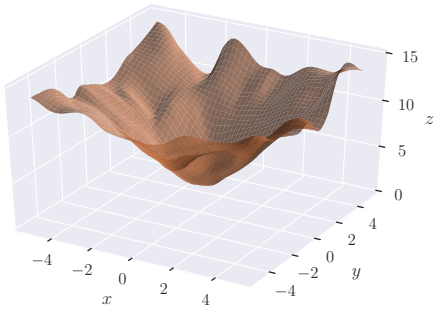
(d) 3×3



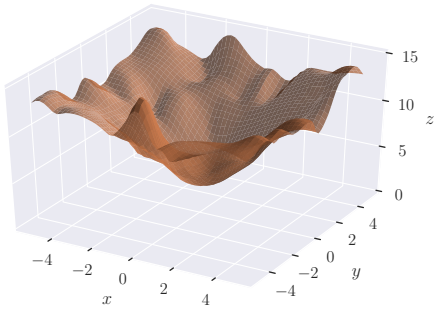
(e) 4×4



(f) 5×5



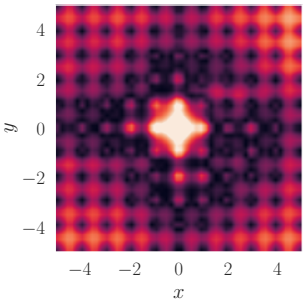
(g) 6×6



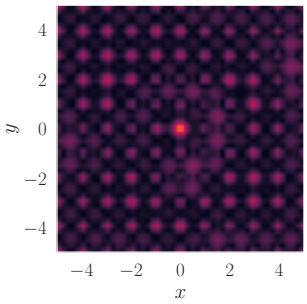
(h) 7×7

Figure 3-9: Reconstruction of the Ackley function by an MLP and different GFNN models. Figure (a) shows the original function.

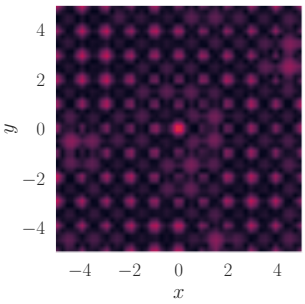
(a) Error scale



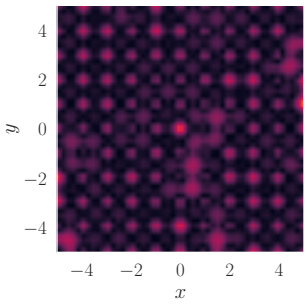
(b) MLP



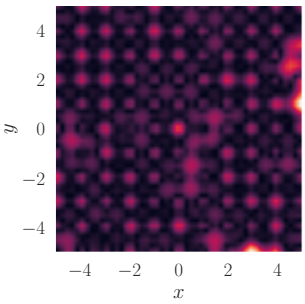
(c) 2×2



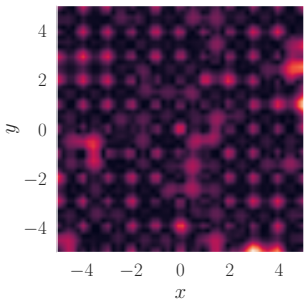
(d) 3×3



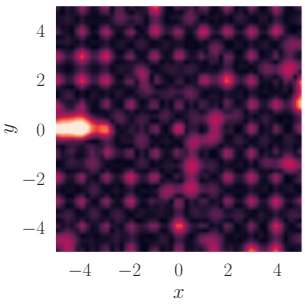
(e) 4×4



(f) 5×5

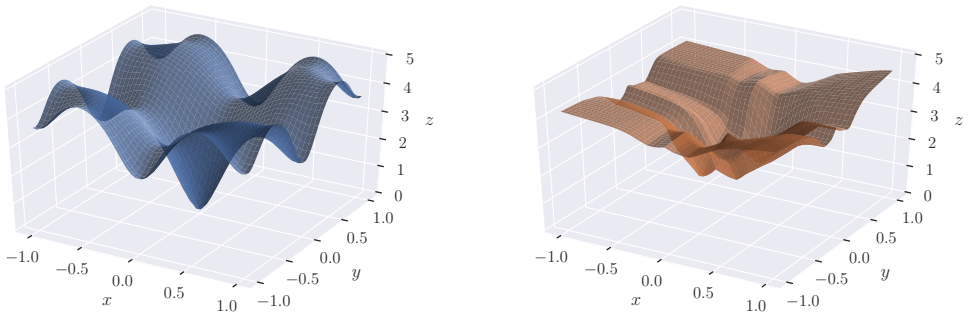


(g) 6×6



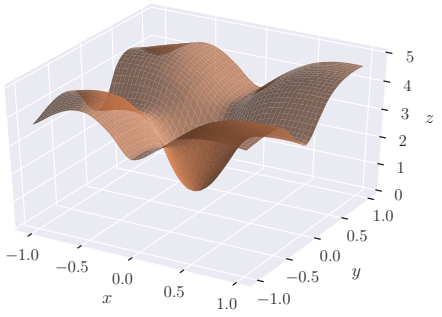
(h) 7×7

Figure 3-10: Error map for the reconstruction of the Ackley function by an MLP and different GFNN models.

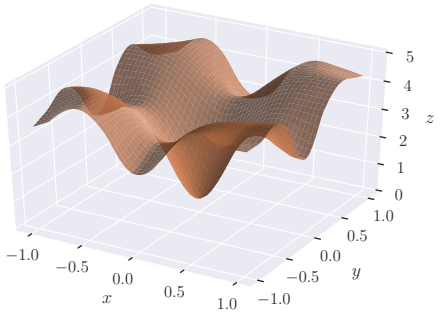


(a) Function

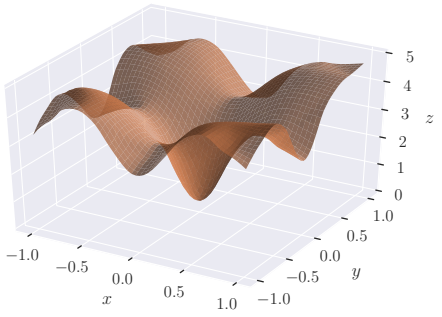
(b) MLP



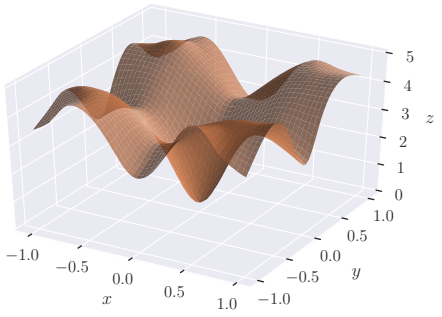
(c) 2×2



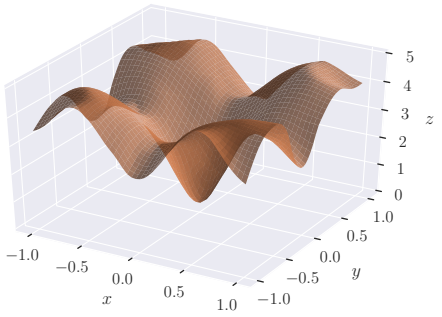
(d) 3×3



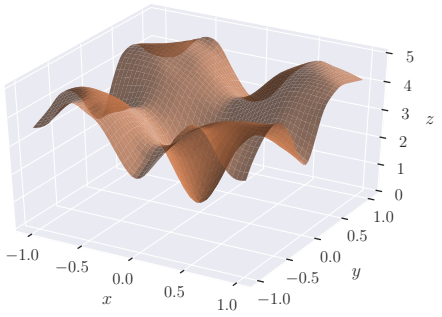
(e) 4×4



(f) 5×5

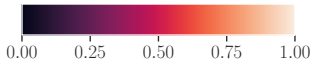


(g) 6×6

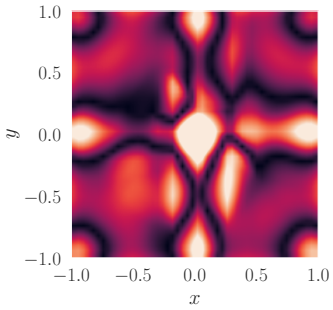


(h) 7×7

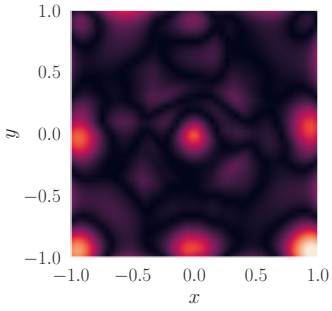
Figure 3-11: Reconstruction of the Ackley function (small region) by an MLP and different GFNN models. Figure (a) shows the original function.



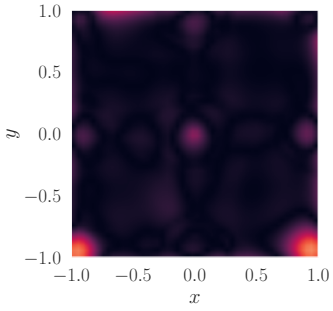
(a) Error scale



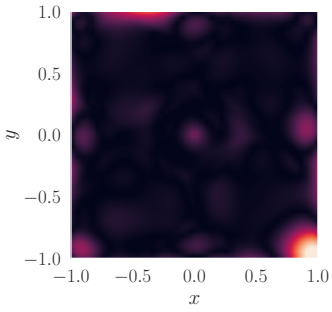
(b) MLP



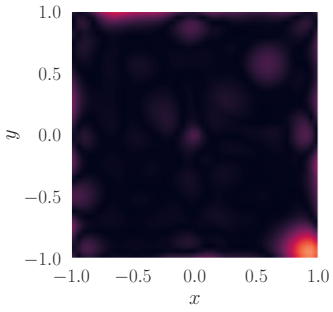
(c) 2×2



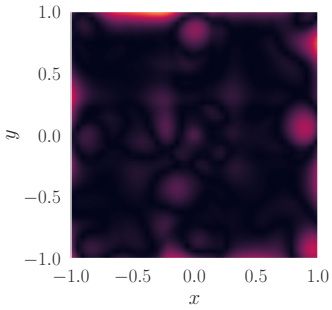
(d) 3×3



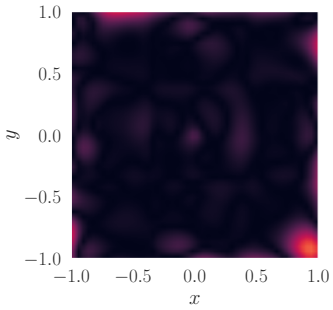
(e) 4×4



(f) 5×5



(g) 6×6



(h) 7×7

Figure 3-12: Error map for the reconstruction of the Ackley (small region) function by an MLP and different GFNN models.

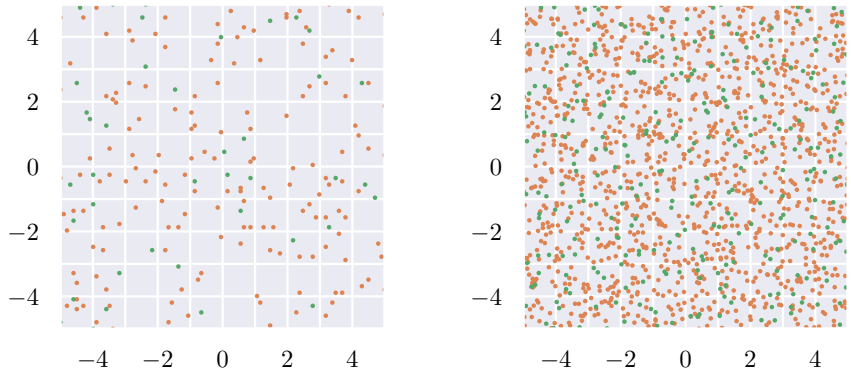
it could be possible at all for a model to learn the shape of this function. From our first results, we can infer that the available data is simply not enough to produce a faithful reconstruction. Upon close inspection of fig. 3-5b (page 37), there are whole hills in the surface for which there is no data at all, so no data-driven model can possibly be expected to reproduce the same shape (unless it is biased towards periodic structures, or it is limited to a particularly well-suited family of functions). With that in mind, we can run a new set of experiments for this problem in particular with a different dataset. This dataset is bigger in size and, instead of being uniformly sampled across the function domain, it is produced through stratified sampling. The goal of this is to ensure that random samples are evenly distributed across the whole domain. For this purpose, this domain is divided into $Q_x \times Q_y$ regions and V random samples are taken from each of these. The complete sample is given by the following expression.

$$\begin{gathered}\bigcup_{i_x=1}^{Q_x}\bigcup_{i_y=1}^{Q_y}\bigcup_{j=1}^{V}\left\{\left[m_x+(i_x-r_x^{i_xi_yj})S_x, m_y+(i_y-r_y^{i_xi_yj})S_y\right]\right\} \\ S_x=\frac{M_x-m_x}{N_x} \\ S_y=\frac{M_y-m_y}{N_y}\end{gathered} \tag{3.16}$$

Where $[m_x, M_x] \times [m_y, M_y]$ represents the domain of the function being sampled and each $\boldsymbol{r}^{i_xi_yj} \in [0,1]^2$ is a random two-element vector with components distributed according to the uniform distribution $\mathcal{U}(0,1)$.

Using this strategy, we can synthesise a new dataset from which it may realistically be expected to infer an accurate model. For this experiment, the domain of the function is divided in 10×10 regions and, from each of these, twelve values are sampled for training and three samples for testing. This makes a total of 1 200 training samples and 300 testing samples. Figure 3-13 shows the distribution of the new dataset across the domain compared with the original one used for the previous experiments.

We know that the first Ackley problem exhibits a complicated shape with a lot of local changes. With a sufficiently rich dataset, a GFNN model with more grid divisions should be able to capture these details. Therefore, for this case we will consider a different set of larger architectures for testing. Recognising that this has been revealed to be a more difficult problem than the other two, it is reasonable to increase the total number of parameters in each model.



(a) Original dataset (b) Revised dataset

Figure 3-13: Data distribution over the function domain in the original and the revised Ackley problem. Green points are training data, orange dots are evaluation data. Grid lines correspond to the hierarchical sampling regions.

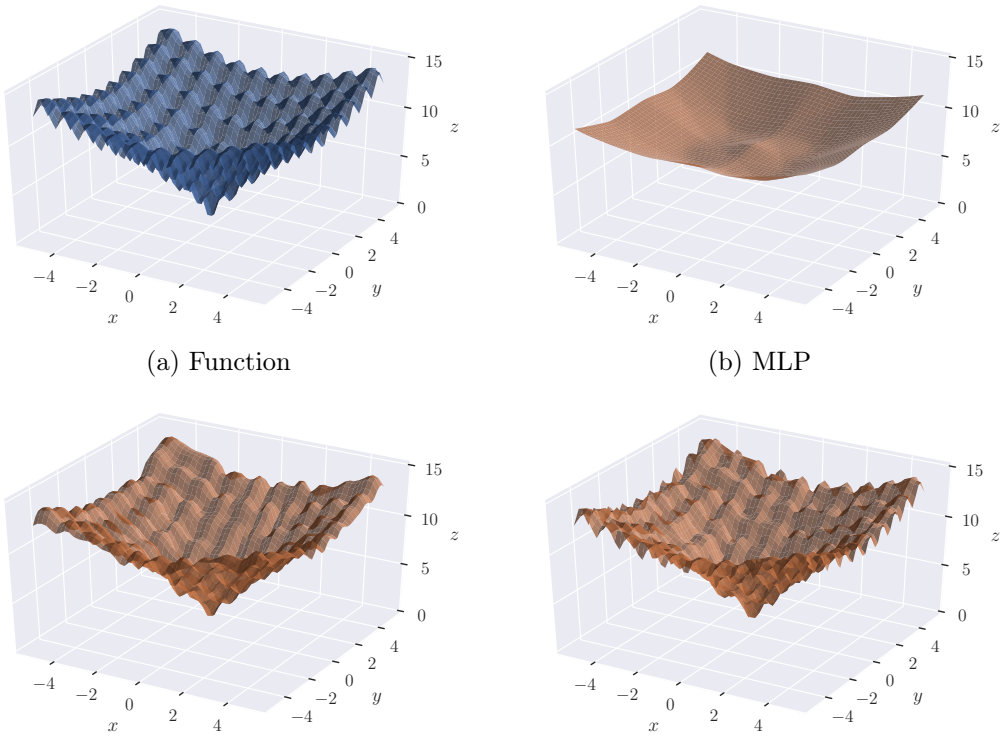
Grid size	Hidden layers	Parameters	Flops	Median	Mean	SD
1×1 (MLP)	250, 250	63 751	190 752	1.556	1.597	0.918
7×7	33, 33	61 495	46 402	0.303	0.380	0.316
8×8	29, 29	63 168	36 494	0.308	0.383	0.307
9×9	25, 25	60 831	27 770	0.244	0.329	0.283
10×10	22, 22	59 500	22 004	0.194	0.289	0.264
11×11	20, 20	60 621	18 530	0.178	0.256	0.257
12×12	18, 18	59 760	15 352	0.172	0.256	0.268
13×13	17, 17	63 375	13 874	0.143	0.233	0.246
14×14	15, 15	58 996	11 140	0.148	0.225	0.233
15×15	14, 14	60 075	9 884	0.158	0.248	0.258
Interp. (training data – 1 200 samples)				0.174	0.248	0.238
Interp. (15×15 grid)				0.446	0.529	0.435

Table 3-3: Evaluated models and error statistics for the revised Ackley problem. Last rows show the error statistics using linear interpolation over the training data and over a 15×15 regular grid of interpolation values.

Table 3-3 shows the tested models and their performance over this new dataset. Arguably, with such a dense dataset training a complex machine learning model seems superfluous. As shown in the table, simply using linear interpolation with the available training data yields better estimates than most models. However, as a test for the learning capabilities of GFNN models, it shows how the architecture can take advantage of the new data and additional parameters much more effectively than the traditional model. Grid models also outperform simple interpolation over a 15×15 grid of interpolated values, showing the benefits of combining nonlienar neural computation with local parameter interpolation. Even with a very small number of units per layer, larger grids are able to produce increasingly better results up to about a grid size of 14×14. Comparing these results to the previous experiments summarised in table 3-2, we can see, for example, that the MLP mean error has only been reduced by about 10%, whereas the smallest GFNN mean error in this set of experiments reduces the previous smallest mean error by more than 60%. The grid models are not only better in absolute terms, but they also exhibit much better scalability and flexibility in relative terms too.

Figure 3-14 shows a visual representation of the reconstructed functions by different models. Clearly, the MLP model is simply not capable of learning the details of the function, and it even struggles to approximate the lowest-frequency patterns in the data. The GFNN model, on the other hand, produces a close reconstruction of the original, comparable to an interpolated reconstruction. The error maps in fig. 3-15 show the nearly identical distribution of the error for both of these reconstructions.

Overall, these results show a vast superiority of GFNN over MLP models over a variety of problems. In all cases, GFNN models have yielded far lower overall errors than MLP models with a comparable number of parameters. Not only that, but GFNN has been shown to be able to scale to more complex problems when sufficient data is provided. The GFNN grid size introduces a new kind of hyperparameter that allows flexibility when deciding how to distribute network parameters throughout the domain of the problem, in turn permitting the model to accurately capture complex local patterns. These results over synthetic evaluation functions are complemented by the results in chapters 4 and 5, where GFNN is successfully applied to more complex scenarios. These evidence the general utility of the model beyond the scope of this evaluation and will show further benefits of our approach specific to these applications.



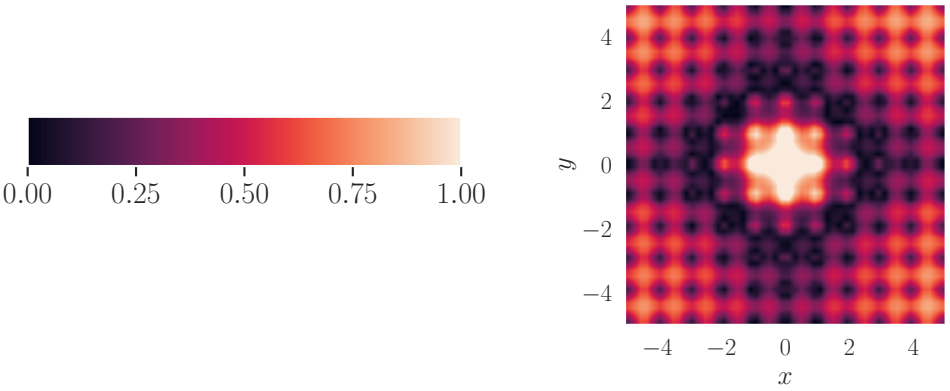
(a) Function

(b) MLP

(c) 14×14

(d) Interp. (training data – 1 200 samples)

Figure 3-14: Reconstruction of the Ackley function by an MLP and a GFNN model trained on the revised dataset, and a linear interpolation reconstruction. Figure (a) shows the original function.



(a) Error scale

(b) MLP

(c) 14×14

(d) Interp. (training data – 1 200 samples)

Figure 3-15: Error map for the reconstruction of the Ackley function by an MLP and a GFNN model trained on the revised dataset, and for a linear interpolation reconstruction.

3.4 Discussion

This chapter introduced the GFNN architecture, a generalisation of the PFNN model that combines a structure of experts across several dimensions of the problem at hand. It makes it possible to learn specialised knowledge for different subspaces of the domain with explicit control over the distribution of the experts and the relationships between them.

As has been shown, GFNN can produce vastly superior results than traditional MLP models in some problems. In addition to learning more detailed local patterns from the data, it also produces better results when the available data grows, exhibiting a more scalable behaviour. It is a powerful model architecture that does not impose any limiting prerequisites, and it is able to yield higher-quality results wherever required within the domain of the problem.

The next chapters make use of GFNN in specific animation synthesis contexts, showing how this architecture can be flexibly adapted to perform competitively in different problems.

Chapter 4

Modelling Quadruped Locomotion

The GFNN architecture presented in the previous chapter can be generally applied to any machine learning problem, but we are interested in its applications to animation synthesis. This chapter describes how it can be used for this purpose, and the specific benefits it can bring to the table.

The problem of quadruped locomotion synthesis was briefly mentioned before as an interesting case that inspired a new kind of neural network model. As a skeletal locomotion problem, it presents a qualitatively different challenge from the bipedal case; while bipedal locomotion obeys a strict cycle of left and right foot alternation, the same cannot be said for quadruped locomotion. As shown in fig. 4-1, the motion pattern of the four legs change with the different gaits the character may adopt. The problem with this is there is no consistent feet cycle in the animation, which is what motivated the design of the MANN model, containing an unstructured set of experts, as a replacement for the PFNN model, featuring a one-dimensional sequence of experts.

Quadruped locomotion is then a perfect testbed to evaluate the animation synthesis capabilities of the multi-dimensional structure of GFNN. Comparing it to MANN, which has already been proved successful on this problem, will give us a reliable assessment of its value.

4.1 Problem Overview

Here we describe the general approach to the problem, loosely following the methodology proposed by Holden, Komura and Saito, (2017) and Zhang et al., (2018). The goal is to produce an effective data-driven locomotion controller for a quadruped character

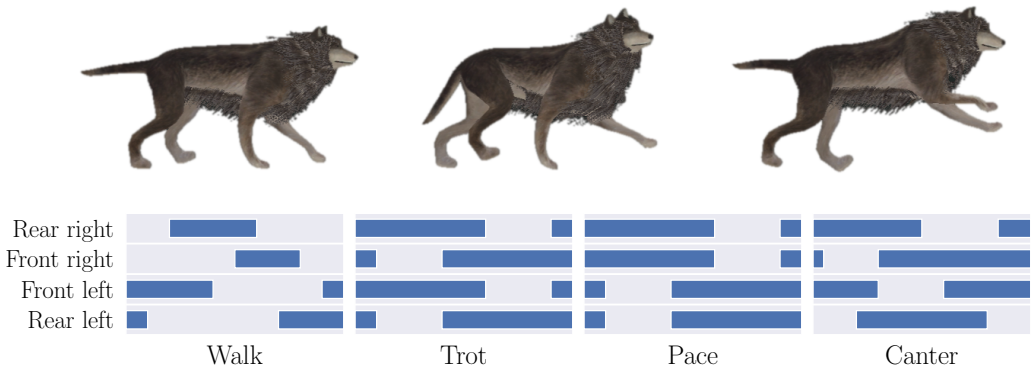


Figure 4-1: Examples of different quadruped locomotion patterns are shown on top. Below, each diagram represents a complete walking cycle for a particular gait, and the blue regions indicate floor contact periods for each foot (Coros et al., 2011).

from a collection of motion capture clips. The controller must be able to synthesise the appropriate animation for the character that satisfies the control input given at any time, maintaining the locomotion style displayed in the available data. For our purposes, we will not consider other kinds of motion like jumping or laying down, which could be integrated by blending particular motion clips into specific points in the stream of synthesised animation. We focus specifically in controlling the direction of the character and the velocity, ranging from a slow-paced walk to a fast gallop.

The controller is modelled as a neural network model trained to continuously predict a new character pose on every frame. The overall scheme is shown in fig. 4-2. Pose synthesis works in an autoregressive fashion: at every step, the information from the current pose is fed to the model along with the control input to produce the pose for the next frame, which in turn becomes the input pose in the next step. The control input represents the "instructions" given to the character, sent from a physical controller or some other means. Obviously, motion capture data does not have any associated control information as such, it is simply performed by the actor and recorded. In order to be able to relate the control signals at runtime with the available data, we use the character trajectory as control input. Specifically, the recent past trajectory and the desired future trajectory. From the point of view of the data, it is trivial to compute the past and future trajectory at every frame. At runtime, though, the future trajectory can be difficult to estimate, as it is influenced both by the requested control and the current animation context. The model can therefore take care of predicting the future trajectory of the animation, which is then combined with the received control signals to form the model control input for the next frame.

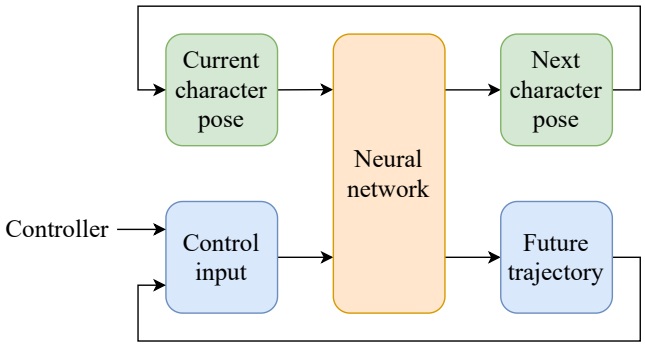


Figure 4-2: Basic neural animation setup.

Zhang et al. showed impressive results on this problem using the MANN model described above. We follow their approach with minor simplifications, replacing only the architecture of the neural network responsible for synthesising the pose, so as to evaluate the potential of GFNN models and their applicability to animation scenarios.

4.2 Data Description

We start from the dataset of quadruped motion capture collected by Zhang et al., which contains clips of different gaits and actions performed by a quadruped skeletal mesh with $J = 27$ joints recorded at 24 frames per second. Here we focus specifically on locomotion. The dataset is a collection of examples of the form $(\boldsymbol{x}^t, \boldsymbol{y}^t) \in \mathbb{R}^{D_{In}} \times \mathbb{R}^{D_{Out}}$, with $t \in \{1, \ldots, T\}$ the number of the corresponding frame of motion data. Each $\boldsymbol{x}^t$ is the pose of the character and the control information at frame t, which is what the model receives as input, while $\boldsymbol{y}^t$ is the pose and future trajectory of the character at frame $t+1$, which the model should predict. Specifically, $\boldsymbol{x}^t = \{\boldsymbol{r}^t_{-T_P}, \ldots, \boldsymbol{r}^t_{T_F-1}, \boldsymbol{d}^t_{-T_P}, \ldots, \boldsymbol{d}^t_{T_F-1}, \boldsymbol{j}^t_1, \ldots, \boldsymbol{j}^t_J, \boldsymbol{v}^t_1, \ldots, \boldsymbol{v}^t_J\}$ is broken down in in four parts as follows:

1. A sequence of $T_P + T_F$ vectors $\boldsymbol{r}^t_i \in \mathbb{R}^2$ representing the planar location of the character at frame $t + iT_S$ with respect to its current location. T_P and T_F are respectively the number of past and future locations included in the trajectory, and T_S is the temporal distance in frames between each pair of locations.
2. A sequence of $T_P + T_F$ vectors $\boldsymbol{d}^t_i \in [-1, 1]^2$ representing the orientation of the character at frame $t + iT_S$ with respect to its current orientation. Orientations are expressed as unit vectors on the horizontal plane.
3. A sequence of J vectors $\boldsymbol{j}^t_i \in \mathbb{R}^3$ representing the 3D location of joint i of the character at frame t with respect to the root.
4. A sequence of J vectors $\boldsymbol{v}^t_i \in \mathbb{R}^3$ representing the velocity of joint i of the character

at frame t with respect to the root.

Similarly, each $\boldsymbol{y}^t = \{\boldsymbol{m}^{t+1}, \omega^{t+1}, \boldsymbol{r}_1^{t+1}, \ldots, \boldsymbol{r}_{T_F}^{t+1}, \boldsymbol{d}_1^{t+1}, \ldots, \boldsymbol{d}_{T_F}^{t+1}, \boldsymbol{j}_1^{t+1}, \ldots, \boldsymbol{j}_J^{t+1}, \boldsymbol{v}_1^{t+1}, \ldots, \boldsymbol{v}_J^{t+1}, \boldsymbol{q}_1^{t+1}, \ldots, \boldsymbol{q}_J^{t+1}\}$ comprises seven parts:

- A vector $\boldsymbol{m}^{t+1} \in \mathbb{R}^2$ representing the 2D displacement of the character root between frames t and $t+1$.
- A scalar $\omega^{t+1} \in \mathbb{R}$ representing the angular change in the orientation of the character between frames t and $t+1$.
- A sequence of T_F vectors $\boldsymbol{r}_i^{t+1} \in \mathbb{R}^2$ representing the planar location of the character at frame $t+1+(i-1)T_S$ with respect to its location at frame $t+1$.
- A sequence of T_F vectors $\boldsymbol{d}_i^{t+1} \in [-1,1]^2$ representing the orientation of the character at frame $t+1+(i-1)T_S$ with respect to its orientation at frame $t+1$.
- A sequence of J vectors $\boldsymbol{j}_i^{t+1} \in \mathbb{R}^3$ representing the 3D location of joint i of the character at frame $t+1$ with respect to the root.
- A sequence of J vectors $\boldsymbol{v}_i^{t+1} \in \mathbb{R}^3$ representing the velocity of joint i of the character at frame $t+1$ with respect to the root.
- A sequence of J vectors $\boldsymbol{q}_i^{t+1} \in [-1,1]^6$ representing the rotation of joint i of the character at frame $t+1$ with respect to the root. Each rotation is expressed as the rotated unit vectors along the x and y axes (see appendix B).

For the purposes of the experiments described in this section, we fix the trajectory size to $T_P = T_F = 6$, with a distance between trajectory points of $T_S = 4$. That is, the model receives trajectory information from one second in the past (24 frames) to 5/6 of a second in the future (20 frames). This implies a total input size of $D_{In} = 210$ and an output size of $D_{Out} = 351$.

4.3 Neural Network Design

As discussed, GFNN models allow us to define a grid of experts across an arbitrary set of dimensions of a problem. We have already seen that the walk phase is not well defined for quadruped locomotion, due to the variety of possible gaits and corresponding feet contact patterns. However, we can still break down the problem into other meaningful aspects. Concretely, if we analyse the components of the control directing the character, it comes down to two basic factors: velocity and direction. We can therefore design a grid model specialised on particular subspaces of this combination of parameters.

Although velocity and direction are not part of the information available to the model as such, it is easy to derive an estimation from the current future trajectory data. This

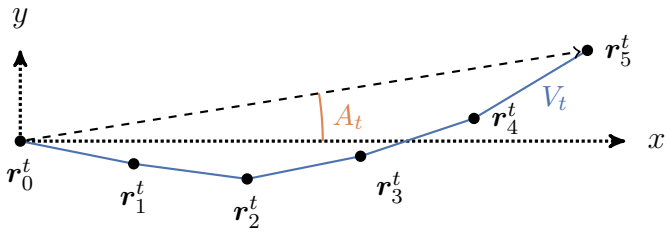


Figure 4-3: Character velocity and direction angle estimation from future trajectory.

is depicted in fig. 4-3. As an estimation for the velocity, V^t, we take the aggregated length of each future trajectory section (in blue), divided by the time span of the future trajectory:

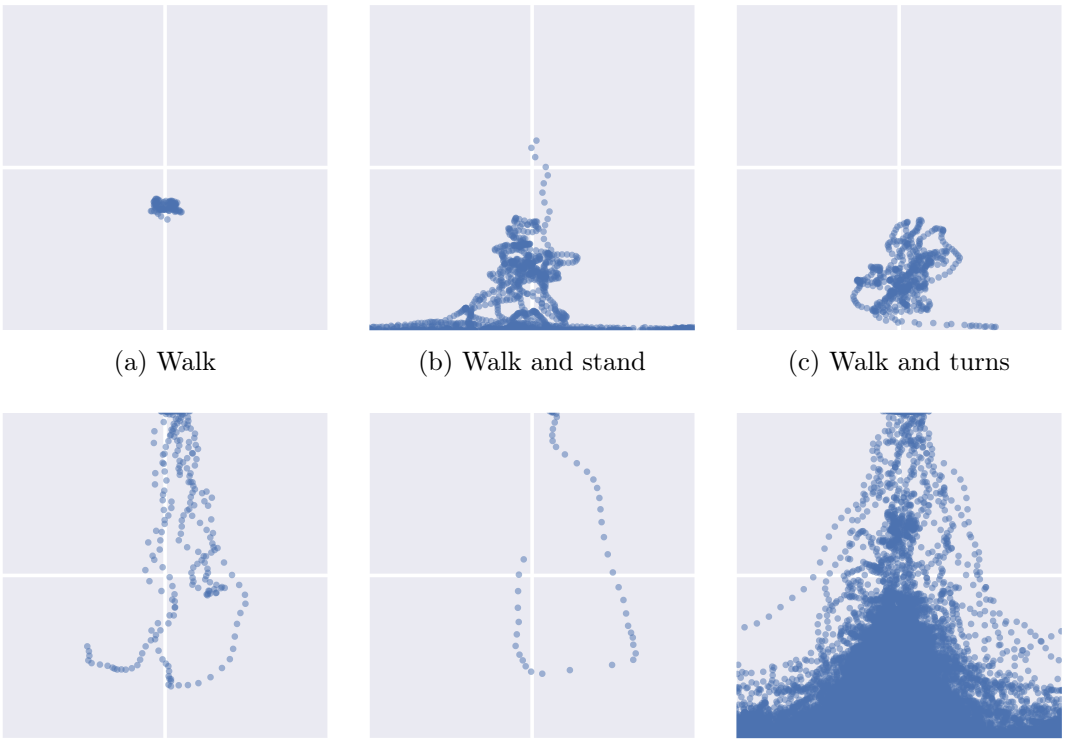
$$V^t = \frac{\sum_{i=1}^{T_F-1} \left\| \boldsymbol{r}_i^t - \boldsymbol{r}_{i-1}^t \right\|}{\delta(T_F - 1)T_S} \tag{4.1}$$

Where $\delta = 1/24$ is the duration of each captured frame.

Similarly, the control direction, $A^t \in [-\pi, \pi]$, expressed as the angle in radians of the requested deviation from the current forward direction, can be estimated as the angle formed by the vector from the current position of the character to the last future trajectory point (orange arc in fig. 4-3). Since positions are already expressed with respect to the current character position and orientation, the forward direction vector, corresponding to zero radians, is always $[1, 0]$ (by convention, the forward direction is the positive x axis), and therefore A^t is given by the angle between the last future trajectory position $\boldsymbol{r}_{T_F-1}^t$ and the aforementioned vector:

$$A_t = \arctan2\left(r_{T_F-1_y}^t, r_{T_F-1_x}^t\right) \tag{4.2}$$

These are good indicators on which to base our model, but they cannot be directly used as grid dimensions. Each example in the dataset must be mapped to a point in $[0, 1]^2$, and both V^t and A^t take values well beyond the unit interval. In order to obtain the 2D grid location $\boldsymbol{\gamma}^t \in [0, 1]^2$ associated with each example $\boldsymbol{x}^t$, V^t and A^t are simply linearly transformed to a normalised value:



(a) Walk (b) Walk and stand (c) Walk and turns

(d) Walk and gallop (e) Gallop and turns (f) All data

Figure 4-4: Distribution of data points across the $[0, 1]^2$ GFNN grid space for different character actions. Horizontal and vertical axes represent respectively γ_1, proportional to the control direction angle, and γ_2, proportional to the velocity (see eq. (4.3)).

$$\begin{aligned} \gamma_1^t &= \frac{A^t + \pi}{2\pi} \\ \gamma_2^t &= \min\left(\frac{V^t}{V_{Max}}, 1\right) \end{aligned} \tag{4.3}$$

Where V_{Max} is an upper bound for V^t set to 300. Figure 4-4 shows how different examples get distributed across the 2D space of the grid. There is a notable negative correlation between γ_1 and $|\gamma_2|$. The reason is that close changes of direction can only really be performed at lower velocities, while high-velocity locomotion only allows for minor trajectory corrections.

The GFNN architecture consists of a 3×3 grid of expert parameterisations. Each expert has two hidden layers with 512 units each with ReLU activation and output layer with linear activation. This is equivalent to the expert architecture proposed by Zhang et al.,

(2018), which also featured two hidden layers with the same number of units.

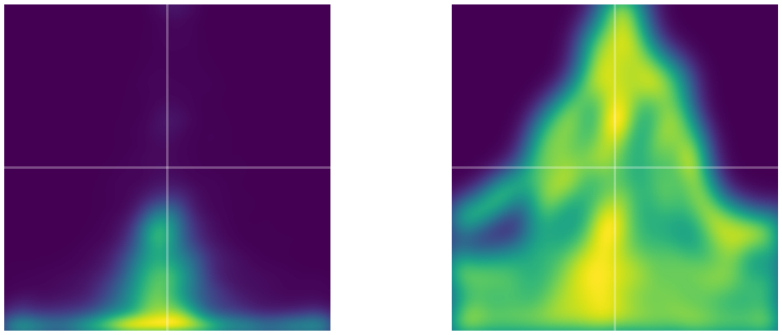
4.4 Training

The available locomotion data consists of about 23 minutes of motion clips, that is, over 33 000 examples, from which 15% (approx. 5 000) are reserved for testing to control for overfitting during training. The training data, however, is not evenly distributed across the spectrum of possible character animations. Specifically, most of the examples correspond to standing and medium-speed locomotion in a straight line, with far fewer examples of turns and high-speed locomotion. This can be clearly seen in fig. 4-4f, which shows how most of the data concentrates in a relatively narrow region of the grid space. The vast majority of the data points have either low speed or small control direction angle, and so the expert parameterisations associated to those regions are much more frequently used than the rest.

This does not suit well the GFNN model, because it means some parts of the grid will be trained far more intensively than others, leading to biased results and unstable behaviour in the outer, emptier regions of the grid. To mitigate this, the training data can be restructured into a distribution that better matches the model architecture. Specifically, to compensate for the imbalance in the dataset, we rearrange the data in such a way that examples from sparser regions of the grid domain (sides and upper half) are selected more frequently than those from the denser regions (middle and lower half). This is done automatically as follows. First, a probability distribution of the data across the grid dimensions is estimated. To do this, a kernel density estimation method (KDE) can be used (Scott, 1992). This method assigns a probability density $KDE(\boldsymbol{\gamma})\colon [0,1]^2 \to \mathbb{R}^+$ to each point in the grid domain according to the expression:

$$\begin{aligned} KDE\left(\boldsymbol{\gamma}\right) &= \frac{1}{TH^2\sqrt{\det\left(\Sigma\right)}} \sum_{t=1}^{T} K\left(\frac{\boldsymbol{\gamma} - \boldsymbol{\gamma}^t}{H}\right) \\ H &= \frac{1}{\sqrt[6]{T}} \\ K\left(\boldsymbol{\gamma}\right) &= \frac{1}{2\pi} \exp\left(-\frac{\operatorname{tr}\left(\boldsymbol{\gamma}^\intercal \Sigma^{-1} \boldsymbol{\gamma}\right)}{2}\right) \end{aligned} \tag{4.4}$$

where $\Sigma \in \mathbb{R}^{2\times 2}$ is the covariance matrix of all the points. The resulting probability distribution is an aggregation of one two-dimensional normal distribution per data point, with a uniform standard deviation proportional to the "bandwidth" value H, here picked according to Scott's heuristic rule (Scott, 1992). With this, we can define a probability



(a) Original training data (b) Restructured training data

Figure 4-5: Probability density of the KDE distributions of the original and restructured training datasets across the grid domain. Horizontal and vertical axes represent normalised direction and velocity respectively.

value $P^t \in [0, 1]$ for each example that is inversely proportional to its corresponding density estimation:

$$P^t = \frac{1}{KDE\left(\boldsymbol{\gamma}^t\right)} \left(\sum_{i=1}^{T} \frac{1}{KDE\left(\boldsymbol{\gamma}^i\right)} \right)^{-1} \tag{4.5}$$

Finally, a new dataset is built taking examples at random from the original one according to their probabilities.

In our experiment, a KDE model is built from the approximately 28 000 initial training examples, from which a new restructured dataset of 30 000 examples is sampled using this method. In this process, about 75% of the training data is discarded, leaving only close to 10 000 distinct examples in total, most of which appear multiple times in the dataset. The effect of this is a more even distribution of the examples across the grid. As can be appreciated in fig. 4-5, while in the original data the amount of examples out of the middle low region is negligible (though they are there, as shown in fig. 4-4f), the restructured dataset maximises the utilisation of the grid domain by the data, which benefits the training of the GFNN architecture.

Once the training data is prepared, the training procedure is otherwise conventional. The optimisation goal for a given example label $\boldsymbol{y}^t$ and the corresponding prediction $\widehat{\boldsymbol{y}^t}$, $L \colon \mathbb{R}^{D_{Out}} \times \mathbb{R}^{D_{Out}} \to \mathbb{R}$, is simply defined as the squared error between the vectors:

$$L(\boldsymbol{y}^t, \widehat{\boldsymbol{y}^t}) = \|\boldsymbol{y}^t - \widehat{\boldsymbol{y}^t}\|^2 \tag{4.6}$$

The model is optimised using Adam optimisation for 300 000 steps on batches of 32

examples, using a learning rate of 0.0001. To mitigate overfitting, a dropout factor of 0.3 is used, as well as an ℓ^2 regularisation factor of 1.0 (see appendix A).

4.5 Evaluation

There are a number of factors to consider when evaluating an animation model, and unfortunately not all of them can be fully captured by simple criteria. In order to have a complete picture, a comprehensive approach is necessary, considering both quantitative numerical metrics of quality and performance and a user study for a qualitative evaluation of the results.[1] The MANN model by Zhang et al., (2018) is a useful baseline for the comparison, considering it was the first to show high quality results for the very same problem under analysis.

4.5.1 Numerical evaluation

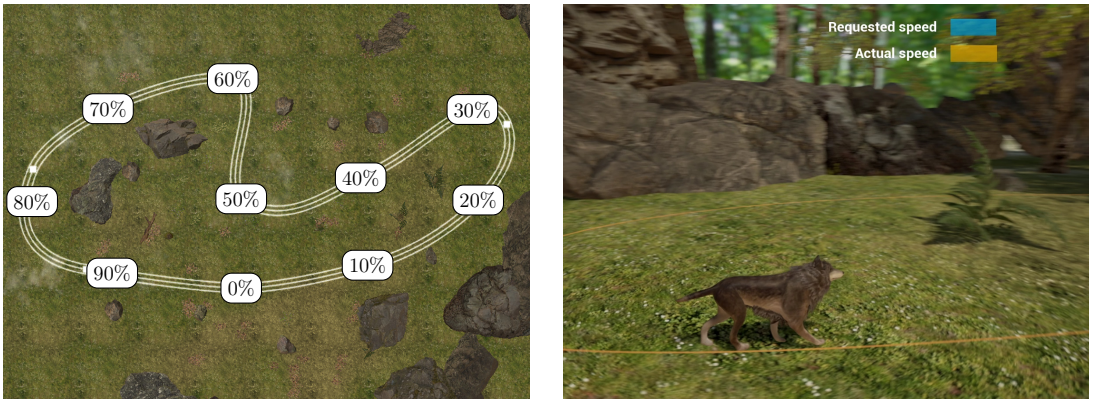
The numerical evaluation covers two aspects of the problem: computational performance and control accuracy. For this, we consider the set of configuration models shown in table 4-1. The "GFNN Cubic 3×3" model is the described grid model based on cubic spline parameter interpolation. The "GFNN Linear" models are different approximations to that trained model using denser grids and linear interpolation, as explained in chapter 3. The "MANN" models, also discussed in that chapter, have a gating network taking a selection of input features with two 32-unit hidden layers, and a pool of experts with an architecture of two hidden layers with 512 units. The "MANN 8" model features eight experts, as proposed by the authors, and is the main baseline considered. The "MANN 9" model uses nine experts, and it is included as an additional comparison of a model with a total number of trainable parameters closer to that of GFNN. As shown, the computational cost of GFNN is higher than MANN, but the linear approximations drastically reduce the number of operations by more than 70% and halve the execution time per frame, giving it a significant advantage over MANN. This comes at the expense of a higher memory consumption and, as discussed later, reduced accuracy. This is a powerful trade-off, as memory is, generally (in the context of real-time animation), a much more abundant resource than CPU time. MANN models, on the other hand, do not offer such flexibility, as their evaluation always requires the mixture of every expert weight.

We consider two factors to measure control accuracy: direction and speed. These provide two clear indicators of the accuracy with which the character follows the given

[1] Code and data available at `https://doi.org/10.15125/BATH-00752` (Dehesa, 2020).

Model	Parameters	Flops	Time (ms)	Memory (MiB)
GFNN Cubic 3×3	4 956 759	12 119 323	7.91	18.9
GFNN Linear 3×3	4 956 759	3 307 237	3.89	18.9
GFNN Linear 5×5	4 956 759	3 307 237	3.85	52.5
GFNN Linear 7×7	4 956 759	3 307 237	3.87	102.9
GFNN Linear 9×9	4 956 759	3 307 237	3.82	170.2
GFNN Linear 11×11	4 956 759	3 307 237	3.75	254.2
MANN 8	4 407 904	9 365 492	5.67	16.8
MANN 9	4 958 688	10 466 994	6.13	18.9

Table 4-1: Model configurations for the numerical evaluation of the quadruped animation. All models use a base architecture with two hidden layers with 512 units each. Grid size of all GFNN models is 3×3. Evaluation time measured on an Intel Core i7-7700K CPU running at 4.20 GHz.



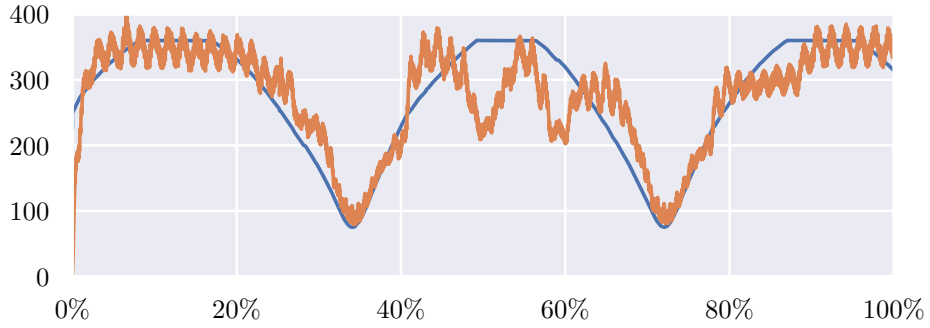
(a) Evaluation path

(b) Evaluation screenshot

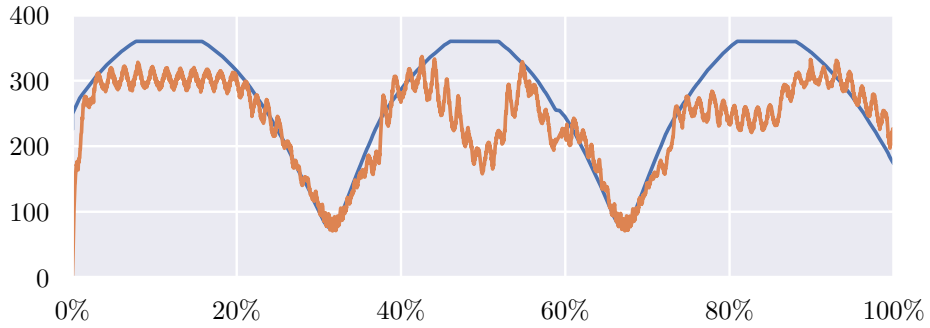
Figure 4-6: Evaluation environment for the quadruped models. Annotations in figure (a) indicate relative progress from the starting point (0%). Figure (b) shows a visualisation of the marked path on the ground and the comparison between requested and actual speed.

control signals. Note the definition of the grid dimensions for our GFNN model closely match these two goals. This is an architectural advantage of GFNN, which allows us to explicitly shape the model towards a specific goal. It also implies, though, that alternative potential evaluation criteria (such as acceleration stability for control smoothness) might benefit from a different analysis and design. In order to evaluate our target metrics, we designed a closed path for the character to follow at a varying speed, shown in fig. 4-6. Comparing the actual position and speed of the character at each frame yields an error metric for the distance from the marked path and the divergence from the requested speed. Table 4-2 shows the error statistics for each of the considered configurations. As explained, the GFNN model is trained on the restructured dataset, but results using the original training data are also included. Conversely, MANN models are trained on the original data, but results with the restructured dataset are presented as well. In general, all models have favourable path error results, as even a mean divergence of about 10 cm has relatively little impact on the control (for reference, the character model size is approximately 90 cm $\times$ 20 cm $\times$ 60 cm). However, the models present very different behaviour in terms of speed divergence. MANN 8 and MANN 9 have both more than double the mean error of GFNN, indicating that MANN does not adapt well to dynamic speed changes. GFNN produces a more reliable result due to the structure of the network being explicitly specialised in each aspect. The different linear approximations produce worse but progressively better results as the number of grid points is increased, as should be expected, and in all cases give a significantly better result than MANN in terms of speed control. Finally, the effect of using a restructured dataset can also be appreciated here. While it improves the results for GFNN, it makes them much worse for MANN, suggesting that the advantage of GFNN is indeed in its architecture, and not in the particular structure of the dataset.

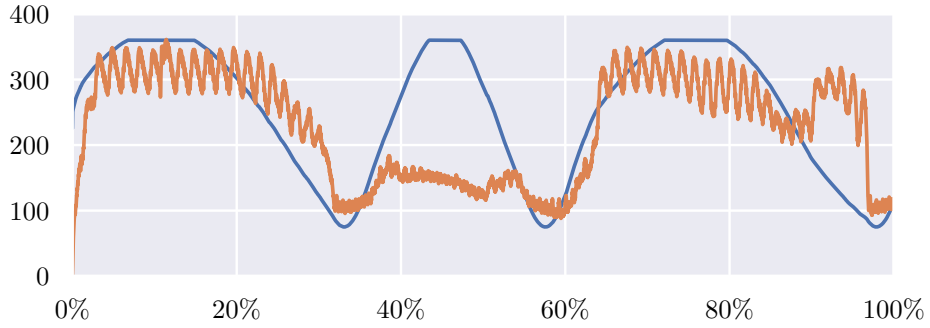
Figure 4-7 gives some more insight into the speed behaviour of the models. Note that, since the requested speed varies with time, its value at each point in the path depends on the actual speed of the character, which is why the blue lines are not exactly aligned in the figures. In general, GFNN Cubic 3×3 maintains a speed very close to the requested value for most of the path. The divergences around 50%, 60% and 80% are explained by the shape of the path itself (fig. 4-6a). Those are points where a high speed was being requested while performing a turn. Figure 4-4 already showed that there is simply no motion capture data fulfilling these conditions. Put in a different way, it is not plausible for the character to complete those turns at that speed. All models, therefore, show significant differences in these situations, which simply reflects the distribution of the available data. The GFNN Linear 7×7 model displays a similar pattern, only accentuating these divergences. However, the MANN models display a



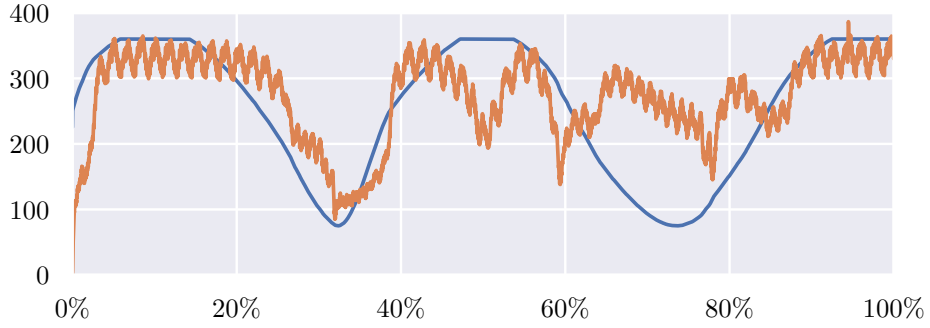
(a) GFNN Cubic 3×3



(b) GFNN Linear 7×7



(c) MANN 8



(d) MANN 9

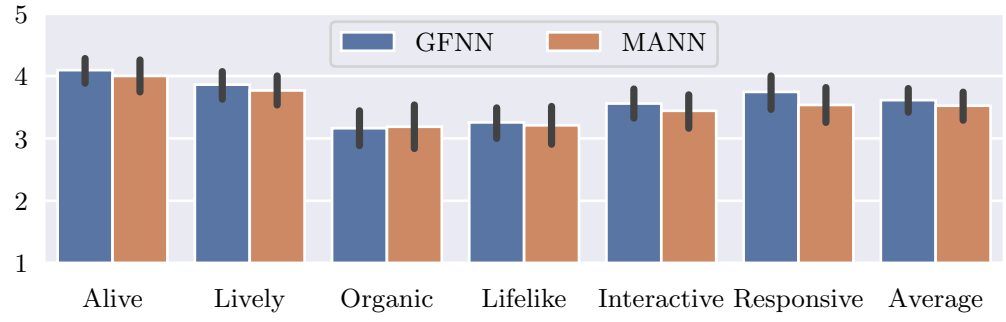
Figure 4-7: Comparison between requested speed (blue) and actual speed (orange) in cm/s for different models. Horizontal axis represents the point in the evaluation path (fig. 4-6a).

Model	Path (cm)			Speed (cm/s)		
	Median	Mean	SD	Median	Mean	SD
GFNN Cubic 3×3	5.7	8.8	8.7	22.5	32.2	29.9
GFNN Cubic 3×3 (no restruct.)	7.7	9.6	7.6	23.6	40.3	41.9
GFNN Linear 3×3	5.0	10.6	11.6	23.9	45.0	51.8
GFNN Linear 5×5	6.5	11.9	11.5	21.1	42.0	48.0
GFNN Linear 7×7	6.0	10.3	10.4	21.2	39.3	43.9
GFNN Linear 9×9	6.2	9.5	9.4	21.5	37.0	40.0
GFNN Linear 11×11	6.0	9.2	9.1	22.1	35.7	37.6
MANN 8	6.6	8.4	8.0	41.3	67.6	63.0
MANN 8 (restruct.)	51.3	46.9	32.9	77.7	82.9	51.3
MANN 9	7.7	10.0	7.8	47.0	65.3	53.9
MANN 9 (restruct.)	38.5	41.1	28.6	73.5	83.9	55.2

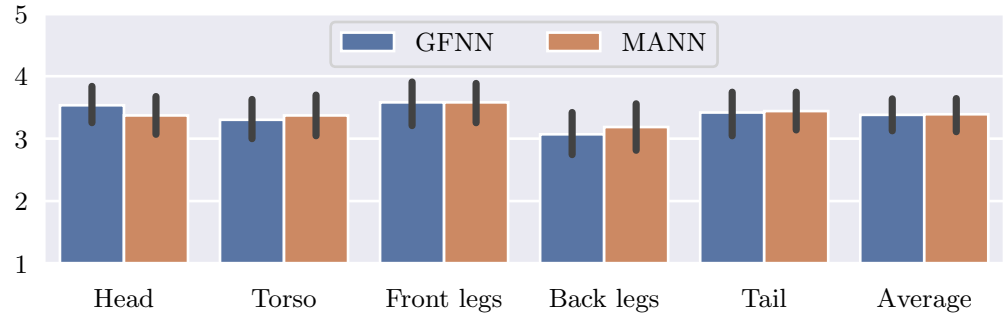
Table 4-2: Error statistics for the evaluated quadruped models.

less predictable behaviour. Both MANN 8 and MANN 9, even when they come close to matching the requested speed, show a noticeable delay (e.g. around 30% along the path). The MANN 8 model fails to transition to a higher speed between in the section between 40% and 60%, and MANN 9 also presents problems in segments that are mostly straight, such as between 60% to 80% along the path. The MANN models appear to have a tendency to get stuck into either a high- or low-speed gait as the requested speed progressively changes, being less responsive to the received control signals.

In addition to these results, it is worth mentioning that MANN also appears to be very sensitive to the random initialisation of the training process. While the results discussed above are representative of an average training run for each of the models, MANN shows far more variance between runs than GFNN does. As an example, after training the MANN 8 model with six different random initialisations, the average mean path error across the models was 8.5 cm with a standard deviation of 1.3 cm, and the average mean speed error was 64.9 cm/s with a standard deviation of 13.9 cm/s. By contrast, the same experiment applied to the GFNN Cubic 3×3 model results in an average mean path error of 9.5 cm with a standard deviation of 0.9 cm and an average mean speed error of 32.0 cm/s with a standard deviation of 1.4 cm/s. This again highlights the advantage of having an explicitly structured mixture of experts, which produces more predictable results than the implicit modelling within the pool of experts in MANN. It is also of interest that the training of all these MANN models converge to a similar loss value, suggesting that the MANN model might need a more complex minimisation goal than the square error at the output to address these issues.



(a) Perceived animacy (higher is better)



(b) Perceived realism across body parts (higher is better)

Figure 4-8: Results of the quadruped animation study. Bar height represents mean value, black lines are 95% confidence intervals.

4.5.2 User study

Unfortunately, the numerical analysis does not actually reveal whether the animation looks realistic and natural or not. The MANN model produces very natural animations for the quadruped characters, and the benefits of GFNN should not come at the expense of this quality. This is, however, difficult to evaluate through objective metrics. For example, a smaller average error in the pose prediction does not entail more natural-looking sequences of animation, as the dynamics of the character motion (for example, a distinct locomotion cycle in the feet) are more important than the precise position of each joint on each individual frame. A user study can provide verifiable feedback on the perceived quality of the animation produced by each model.

We prepared two evaluation videos showcasing the behaviour of GFNN and MANN through the circuit shown in fig. 4-6 (page 62). Specifically, the GFNN Linear 7×7 and MANN 8 models were chosen as evaluation conditions. MANN 8 is the most competitive of the two considered MANN models in the numerical evaluation, while GFNN Linear 7×7 offers a good balance between quality, size and computational cost, lower

than that of MANN 8. We designed a study where participants were shown each of the videos (in random order) and were asked to complete a questionnaire about their impressions on them. The first part of the questionnaire was a set of semantic differential scales from the "Animacy" section of the Godspeed questionnaire (Bartneck et al., 2009), initially designed for robot perception evaluation but applicable to our context. This includes five-point scales for the pairs of adjectives "Dead / Alive", "Stagnant / Lively", "Mechanical / Organic", "Artificial / Lifelike", "Inert / Interactive", "Apathetic / Responsive". In addition to that, participants also had to evaluate the realism of the animation of each part of the character, including head, torso, front legs, back legs and tail, using a five-point scale graded from "Very unrealistic" to "Very realistic". After the questionnaire for each of the conditions, a final five-point scale asking which one of the two videos was more realistic was included, with one being MANN and five being GFNN. The study was completed by a total of 43 participants, aged from 19 to 54 years (mean 30.0, s.d. 9.5) and including 8 females, 32 males and 3 participants of unspecified gender. On scales from one ("No experience") to five ("Expert"), participants reported an average experience in computer animation of 2.09 (s.d. 1.04), in video game development of 2.44 (s.d. 1.37) and a general experience with video games of 3.65 (s.d. 1.11).

Figure 4-8 shows the results of the study. Overall, it is clear that the user perception is nearly identical for both models. GFNN appears to receive a slightly more positive evaluation in some aspects, but not to a statistically significant degree. To the final question of which of the conditions seemed more realistic, the average response was 3.14 (s.d. 1.46), also suggesting very evenly distributed ratings. A more detailed statistical analysis can be found in table 4-3. Here, two t-tests are performed: a lower-bound equivalence test and a two-tailed test. The first test evaluates the hypothesis that GFNN ratings are not lower than those of MANN. The second test evaluates the hypothesis that the ratings from GFNN differ from those of MANN.

The equivalence testing method requires the selection of an equivalence region value, which is the threshold below which differences in the sample means are considered insignificant (Dixon et al., 2018). We picked an equivalence region 0.2, which represents a 5% of the score range in a five-point scale, and is consistent with the observations of other authors about this kind of questionnaire (Syrdal et al., 2013; Kartsidis et al., 2014). It also corresponds to a Cohen's d of less than 0.2 over the standard deviation of 1.02 of the overall set of ratings, which constitutes less than a small effect size (Cohen, 1977).

The results show that, in terms of animacy, GFNN is at least as positively rated as

Scale	GFNN $\not<$ MANN		GFNN $\neq$ MANN	
	$t(42)$	p	$t(42)$	p
Animacy				
Dead / Alive	2.56	0.007*	0.81	0.421
Stagnant / Lively	2.46	0.009*	0.78	0.439
Mechanical / Organic	0.90	0.185	−0.12	0.906
Artificial / Lifelike	1.43	0.081	0.27	0.789
Inert / Interactive	2.36	0.011*	0.87	0.390
Apathetic / Responsive	2.78	0.004*	1.42	0.162
Average	2.60	0.006*	0.80	0.427
Realism				
Head	2.82	0.004*	1.27	0.212
Torso	0.81	0.212	−0.43	0.667
Front legs	1.31	0.098	0.00	1.000
Back legs	0.47	0.319	−0.66	0.514
Tail	1.20	0.118	−0.16	0.875
Average	1.81	0.039*	−0.09	0.930

Table 4-3: Analysis of the quadruped animation user study. Starred values indicate statistical significance ($p < 0.05$).

MANN with high significance. The realism scales, on the other hand, are not as conclusive, with several p-values in the equivalence test well over 0.05. However, in no case do the two-tailed test support the hypothesis that the ratings for GFNN and MANN are different either. The average value of the realism scales, though, does show a significant result, suggesting again that the animation from GFNN is not perceived as less realistic than the animation generated by MANN. Comments from the participants also indicate the differences between the models are minor, with several contradictory assessments between respondents. Adjectives like "stiff", "mechanical", "unrealistic" or "static" were used to describe GFNN in sixteen occasions and MANN in fourteen, while words like "good", "realistic", "convincing" or "organic" described GFNN for fourteen participants and MANN for thirteen. Some comments addressed the control issues analysed above, with the speed for MANN feeling "fairly delayed" with respect to that requested, while GFNN seemed to "match the requested speed and path more often". In any case, overall perception from the participants was generally positive, noting the videos were "pretty good overall", and even though the animation seemed "a little stiff" for some participants, it also looked "amazing" for others.

These results confirm that the quality of the animation produced by GFNN is at least as good MANN, yet it is less computationally expensive and provides more precise control

over the character speed. It also offers more flexibility in balancing time, memory and accuracy, as well as better predictability over the results of the training process.

4.6 Discussion

This chapter has shown how a GFNN model can be used in an animation synthesis problem, and particularly in the context of quadruped character locomotion. The flexible architecture of GFNN allows us to decide how to approach the problem, selecting the aspects of the data that we consider more relevant to specialise the model. Here, the key dimensions were chosen to be directly proportional to the direction and velocity of the character, which makes the structure of the model simple and easy to understand. The presented dataset restructuring methodology also showed that it is feasible to train a grid model with very unevenly distributed data across its dimensions.

With respect to the quality of the results, user evaluation confirms that GFNN rivals the state-of-the-art MANN model in realism and animacy while offering better performance, flexibility and responsiveness. Importantly, GFNN resulted in significantly more predictable results, both in terms of sensitivity to training initialisation conditions and runtime behaviour. This highlights the benefits of an explicit model structure, where the specialisation of experts is predetermined and unambiguous, in contrast with the implicit structure of MANN.

Our results show the potential of GFNN as a model for animation synthesis. In the next chapter, this architecture is applied to a different but comparable animation scenario, as part of a larger interaction system.

Chapter 5

A Framework for Human–Character Interaction in Virtual Reality

The creation of virtual worlds populated with lively characters is one of the main crafts involved in the development of video games, and a careful design of those characters is key to the believability of that world. In most cases, when creating a virtual interactive character for a video game or a similar medium, one does not start from an explicit theoretical model for the interaction. Instead, the underlying conceptualisation evolves organically in an iterative process, conditioned by the kind of the interactions under consideration, but also by the experience of the designer, the available tools and other design choices. For example, the interaction system of a melee combat game may go through several stages where different mechanics, like attacking, defending or evading, are represented in different ways and with different relationships among them. This is a natural process for developing such a complex system, and, to some degree, the specific implementation of that system will necessarily be tightly coupled to the specific interaction that is being modelled. That is, to continue with the same example, if a counter-attacking mechanic is later introduced, its relationship to attacking, defending and evading will need to be explicitly outlined.

However, as technology advances and interactions become more ambitious, an increasing number of issues and concerns that are intrinsic to the medium, more than the specific interaction itself, begin to repeatedly arise. VR is a good example of this. The natural interface offered by motion controllers enable a whole new range of interaction possibilities, and it nowadays fairly straightforward to setup a virtual scene that can be experienced through an affordable VR kit and situate a character in it. However,

if one attempts to recreate even relatively simple interactions between the player and the character, fundamental difficulties quickly emerge. Consider the case of a simple handshake. How can we tell whether a user wants to initiate a handshake? At what point should the character begin to move the hand towards the player? When should the hand be pulled back? What if the player appears to initiate a handshake but then stops the action? It is very difficult to manually implement the exact logic for each of these situations, let alone a more complicated interaction. These issues are inherent to a medium where the player is essentially unconstrained in terms of the motions that can be performed, unlike the traditional case where the avatar is limited to the finite set of possibilities enabled by the designer. It seems clear that there is some unresolved essential complexity in the design of interactions in VR, and constantly having to conceive partial, ad hoc solutions for each particular scenario is simply not practical.

Data-driven methods offer a promising path towards a more effective methodology. Motion data capturing is now a relatively affordable technology that is routinely used by video games. It is much more economic to capture a rich library of motion data than it is to painstakingly author animation clips of comparable quality by hand (Mohd Izani, Eshaq and Norhan, 2003). Given the ability to produce arbitrary amounts of motion data at a low cost, it only makes sense to take advantage of methods that exploit the scale and variety present in large collections of data. We reviewed in chapter 2 some recent successful proposals, and chapters 3 and 4 described a mathematical model that we are introducing for this purpose. This chapter studies how these kind of methods can be coherently integrated into a larger framework that addresses the specific design needs that developers may have. But first, let us analyse the reasons that make this a necessity.

5.1 Complexities of Interaction in Virtual Reality

New media always come with new possibilities, but also with new challenges. Consider for example the leap from 2D to 3D graphics in video games. For all the innovation that this enabled in terms of design and fidelity, it turns out that interacting with a 3D environment using a conventional controller, mouse or keyboard is frequently far less intuitive than with a 2D world. This is mainly due to the complexities of 3D camera control and the two-dimensional nature of these controllers, which sparked a variety of control schemes, each with its own limitations (Jankowski and Hachet, 2013). Nowadays, both consumers and developers can rely on a set of common patterns and conventions that have been progressively established through time. For example, most video game controllers today feature two joysticks, and most games assign character control to the left

one and camera control to the right one. All parties involved (hardware manufacturers, game developers and players) are aware and comfortable with this, for it has become a settled practice. The current situation with VR is comparable, but at an earlier stage, with a growing industry that has not yet arrived to that kind of conventions. We shall therefore start by identifying what is in this medium that makes it fundamentally different to what we already know about interaction with virtual environments.

The first obvious problem is in the nature of the control itself. Control in screen-based interactive media is typically realised through a handheld controller device with directional inputs and buttons. Any action that the player may perform must be expressed through it, and this limitation facilitates the interpretation from the point of view of the designer. For example, if some button is associated with "attacking", each press is a discrete, unequivocal event indicating that this action must be initiated, possibly followed by some reactions in turn. However, this is not how interaction works in VR. While VR controllers do have buttons that could be used to signal player actions (and are used in that way by some titles), the expectation is that the player will be physically performing the intended action. This makes for a far more natural and immersive experience, but obviously becomes much harder to process too. In most cases, the input data from the player comprises the position and orientation of the headset and both hands or, more precisely, hand controllers (features like velocity and acceleration can be considered derivative of these). These positions are not referred to a particular body reference location (e.g. the hips or the feet), since there is no further information about it, but to some arbitrary point in the virtual world. Ideally, when the user performs an action, say attacking, there should be a reaction to it, such as a defence from the opponent. However, it is usually very difficult to reliably determine whether the player is actually performing the action or not. For example, if we only consider to be valid actions those performed along a specific trajectory of the hands, there will be many unsuccessful attempts from the user due to minor execution imperfections, resulting in a frustrating experience. However, too lax identification will result in spurious action recognition events in the system, with the consequent confusion for the player. Moreover, maybe some actions cannot be characterised by proximity to a particular trajectory, and the underlying pattern is not easily expressible in geometrical terms. Hence, the interpretation of the input is the first hurdle when designing an interactive system in VR, because it is not composed of discrete, unambiguous events but of a continuous stream of spatial information.

The second problem lies on the reacting side, and in particular on the animation. Focusing now on interactions with virtual characters (as opposed to interactions with a

static environment), these are expected to react naturally to the actions of the player, usually with some motion. Again, in traditional screen-based experiences this is relatively simple. Since the timing of each event is exactly known, we can control the precise moment when a reaction takes place with respect to the initial action, and how they interact. For example, if a player starts an attack against a character, and the character in turn should react with a defensive action, it can be enacted in a way that both are perfectly matched (such as the character quickly blocking the incoming attack halfway its trajectory). The system is in control of all the motion, both the character and the avatar controlled by the player. A lot of effort can go into making the reaction more believable, polishing the animation, timing and other details, but there is no uncertainty about the development of events. Obviously, this is not the case in VR. Even assuming we know that the player is starting an attack, the character still has to come up with some reaction that matches the exact situation at hand. Unlike fixed animation clips, the attacks from the player will never be exactly the same, they may vary in trajectory, speed and other aspects, and so the reaction cannot be exactly the same either. Also, the character must now be able to handle other kinds of situations, like incomplete actions from the player. All this makes it difficult to build an animation system for the character in the traditional sense. Even having multiple possible reactions for different variations in the input, it is just infeasible to cover every possible case, and it is not an scalable methodology (for example, adding a new set of reactions to the character for a different state, like being "tired", would require reanalysing and reworking the whole system again). So the production of reasonable reactions is the second big challenge to create an interactive character in VR, since there is no predetermined set of possible inputs, but rather a varied spectrum of variations of the same actions.

This chapter presents a conceptual framework that specifically addresses these two issues, providing a general design methodology that is then exemplified in a concrete interaction scenario.

5.2 Framework Overview

In spite of the challenges posed by interactive VR, designers would still like to be able to define the nature and rules of the interaction in the same fashion as they would in the screen-based case. We just need an intermediate layer that, so to speak, takes care of these additional complications in an automatic way, enabling this working methodology. To this end, we introduce a general framework for human–character interaction in VR that uses data-driven models to abstract the low-level details of the interaction, offering designers a conceptual view of the problem about which they can easily reason and

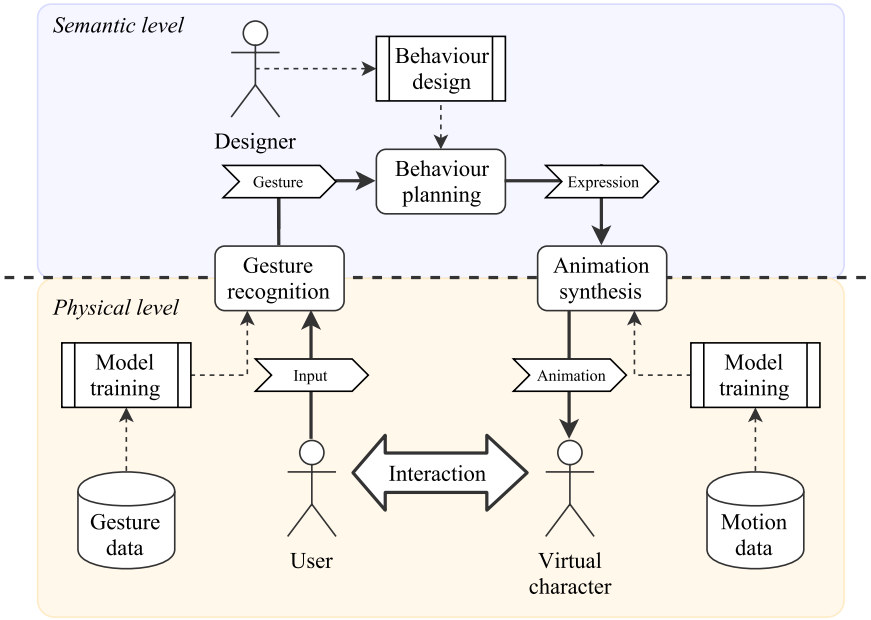


Figure 5-1: Framework diagram

construct the interaction rules. The goal of the framework is to take advantage of the automatic data analysis capabilities of trainable models, assumed as black boxes, while maintaining full control over the character's behaviour. This is an essential need from the point of view of design, and is the reason why we cannot simply use one data-driven model (or an ensemble of them) to drive the interaction. A careful integration of these models with conventional logic can both simplify the problem empower the designer. Figure 5-1 shows a diagram of the framework. It splits the problem into two different levels, namely the *physical* level and the *semantic* level, described in detail in the following sections.

5.2.1 Physical Level

The physical level represents the direct interaction of the framework with the environment. In this sense, "physical" refers to the raw information about the virtual world, mostly of the geometrical kind. This includes, on the one hand, the control signals from the user and, on the other hand, the physical configuration of the character, plus other relevant environmental information. User control is captured by the VR hardware controllers, which, for our purposes, will be assumed to afford spatial tracking of the head and both hands. VR controllers typically have other kinds of input mechanisms, like buttons and joysticks, but we will not be considering these here, as they should not typically take part in a natural interaction with a character. The configuration of

the character is understood as the location and pose of a skeletal structure driving the 3D representation of the character, as is standard in the vast majority of animation methodologies.

The physical level is responsible for two tasks: interpreting the actions of the player and constructing reactions to them. This is done through two corresponding subsystems, namely the gesture recognition module and the animation synthesis module. These modules serve as an interface between both levels, acting as "translators" between low-level information and meaningful labelled data, and both of them are built from previously collected data, without active participation of animators or designers (from the point where the data is available). The gesture recognition module ingests the input from the player and produces a label indicating the gesture that is currently being performed, which is sent to the physical level. In exchange, the animation synthesis module receives another label indicating the action that is expected to be performed by the character, and generates a sequence of character poses accordingly.

The processing done at the physical level is independent of the design of the character behaviour. Within a vocabulary of gestures and character actions, the way each of them is physically performed is predetermined. The pattern identifying a gesture from the player does not change depending on the effect of that gesture, and the way an action should be executed by the character does not depend on the reason why the action is requested. For this reason, having purely data-driven models at this level does not interfere with the design of the character behaviour, and there is no need for transparency or configurability in these modules beyond the ability to define and select the training data used for each one.

5.2.2 Semantic Level

The semantic level aims to simulate the environment in which a designer would work with traditional screen-based media. Unlike the physical level, the information here does not directly correspond to any individual data point in the virtual world, but to the abstractions provided by the data-driven models. The objects are not, therefore, vectors and rotations, but discrete labels with concrete semantics. On the one hand, there are the recognised gestures served by the gesture recognition module, on the other hand, the expression labels indicating the action requested from the animation synthesis module.

This level is owned by the human designer, and there is no automatic learning of any kind. Instead, the information follows predictable, hand-designed rules that enforce the

intended interaction envisioned by the designer. This is not because it is not possible to use machine learning to process information at this level, but rather a deliberate choice to guarantee the complete control by the designer. It contains a single subsystem, the behaviour planning module. Its mission is to interpret the sequence of gestures detected on the input and produce the sequence of expressions that must be realised in response. This is what ultimately characterises the interaction between the user and the character, and so it is important that it is encoded by unambiguous logic that a human can easily understand and refine to meet specific criteria. In principle, the implementation of the behaviour planning module can take any form ranging from simple if/then rules to sophisticated cognitive models. In practice, though, most cases require a behaviour model that is capable of moderate complexity but also easy to use and change. State machines are one of the best established tools in the industry that fits these needs, being reasonably expressive but also well understood by designers (Yannakakis and Togelius, 2018).

As the behaviour planning module is explicitly developed by a human, the framework combines the benefits of automated data-driven processing at the physical level with the transparent and predictable behaviour enforced by the semantic level.

5.3 Case Study: Sword Fighting in Virtual Reality

To reify these concepts into something more concrete, we picked a specific interaction scenario to demonstrate how each of the different components can materialise in a practical case. This should make clearer the role of each element in the whole system and the relationships between the modules. It will also serve as a testbed to evaluate the framework as a complete and coherent interaction system.

The case study selected for this purpose is a VR sword fighting scenario. This encompasses all the features and challenges addressed by our framework. It includes an organic interaction between the player and a virtual character, happening in close distance and with no strict rules or predefined set of "valid" actions. It is as well an appealing kind of experience for VR that can be controlled in a natural way with a standard hardware kit and has obvious mechanics. In simple terms, the result should just be an environment with a virtual opponent capable of attacking and defending from the player with a sword.

Being more specific, we will be considering the case of two-handed sword fighting and an opponent with three skills: blocking incoming strikes, issuing strikes against the player and counter-attacking strikes. To block an incoming strike, the character needs

to identify the player action and put its own sword in the path of the strike. While this could seem like a purely geometrical problem, it is in fact very difficult to predict the speed and trajectory of the player's sword, both due to the noisy signal from the VR controller and the irregular motion of the player's hands. And even with that information, it is not obvious how to produce an appropriate blocking animation from a collection of motion capture data that only covers a limited number of possible sword interactions. The attacks are, in a sense, simpler, since, unlike the reactive blocking skill, they are proactively initiated. The difference between an attack and a counter-attack is that the counter-attack is an immediate response to an attack from the player, and aims to take advantage of the opening that the player may have left in their defence. This requires an understanding of the specific direction of the attack, which will be used to elaborate that response.

Throughout the rest of this chapter we will present each individual part of the framework, on a general level first and then in the context of the specific needs of the sword fighting case, including concrete results for each component and the system as a whole.

5.4 Gesture Recognition

The gesture recognition module is the entry point to our framework. Its purpose is to identify discrete actions in the input given by the player, classified within a fixed vocabulary of gestures known by the system. Having a fixed vocabulary does not imply that the player is expected to perform the actions following a precise pattern, but rather it is a semantic categorisation of the possible interactions that will be useful later for reasoning. For example, a cooperative scenario could have a vocabulary of gestures including *Hello*, *Come here*, *Go right* and *Go left*. What makes each of these gestures is not formally stated in any explicit form. Instead, it will be the collection of gestural data where these are performed that will determine what is and what is not a gesture. From a designer's point of view, defining the gestures comes down to simply "showing" how they are performed.

Unlike many other gesture recognition systems, here the gesture identification is not just provided when the gesture has been completed. The reason is that this may already be too late to build a proper reaction, and introduces a discontinuous state of awareness about the actions of the player in the character. In some cases that may not be a problem, but sometimes it can lead to an unnatural interaction. Following with the previous example, if the player waves a hand to say hello, it may be more natural for the character to wave back soon after the gesture is started than waiting until the player has

finished waving and has the hand down again. For this reason, the gesture recognition module provides instead a continuous stream of predictions on every frame, indicating the gesture that the player is currently performing, or a special *No gesture* label if there is no player activity. This introduces, however, a new need, since reacting as soon as the first frame of a gesture is detected may also be too early. For example, if the player gestures *Come here*, the character should probably be expected to start moving when at least half of the gesture is completed to emulate a human-like reaction time. Gestures can have different duration and be performed at different speeds, though, so simply adding a fixed delay between the first detection and the response is not enough. To overcome this issue, the gesture recognition module provides a second piece of information with every prediction in addition to the detected gesture itself, which is a progress indicator. This indicator takes a value between zero, meaning the beginning of the gesture, and one, meaning its end. So, an output pair from the gesture recognition module like (*Hello*, 0.3) would indicate that the player is performing the *Hello* gesture and it is currently 30% completed, which gives designer a higher level of control when deciding how to interpret this information in each case. This approach provides the framework with a complete view of the actions of the user, and can be easily adapted for a broad variety of scenarios with long, short, static or dynamic gestures.

5.4.1 Sword Fighting Gestures

In the case of sword fighting, the relevant user actions to identify are not hand signs or indications, but the actual sword attacks against the opponent character. We consider eight possible kinds of sword actions, corresponding to eight strike directions: *Up*, *Up-right*, *Right*, *Down-right*, *Down*, *Down-left*, *Left* and *Up-left*. This number represents a compromise between having gesture labels that accurately represent the player action and having a manageable vocabulary size. It is reasonable to adopt these eight directions as most sword strikes can be clearly assigned to one of these categories. If we were to consider, for example, sixteen directions, it would be difficult for most players to consistently tell the difference between contiguous directions, like *Down-right* and *Down-down-right*. This is important because, as described below, the training of the system depends on producing a quality dataset for each of the gestures, so there must be no ambiguity in the vocabulary. We do not consider here more advanced actions like a stabbing attack, which are more difficult to execute effectively, specially in close range sword fighting.

5.4.2 Data Collection

As with any data-driven system, the gesture recognition module can only be as good as the data used to train it. In most cases there will be no readily available dataset that covers the selected vocabulary of gestures for the scenario under consideration, and even if there is, it is unlikely that it will be directly usable for this purpose. For example, a full-body motion capture data clip of sword fighting will most definitely contain sword strikes, but each of them still needs to be individually delimited and labelled for a model to learn from it. Motion capture data also typically contains many more skeletal features that are not relevant in this context (since the VR hardware can only track head and hands) and that require a certain amount of manual polishing after capture. Also, the features that are relevant do not necessarily match exactly their VR counterparts, since the motion capture markers and the VR controllers do not share the exact the same position and orientation.

For these reasons, it becomes necessary to design an effective method to collect gestural data that is simple, flexible and minimises the amount of manual work. We introduce a data collection VR environment built for this exact purpose. This is a VR scenario, independent of the actual scenario being modelled, where an individual can be placed for the purpose of recording gestural data. Figure 5-2 shows an image of this environment. The participant is shown a series of gesture signs (which can be the textual name of the gesture or an iconic representation), and is asked to perform them in sequence. On each gesture, the participant is instructed to start pressing a button in the controller at the beginning of the gesture and release it at the end. This is, obviously, not something that will be required in the actual environment later, as the player should not be expected to notify the system about each strike with the press of a button. But, for data collection purposes, the button can be used to delimit the boundaries of each gesture. Hence, the produced data is automatically labelled without further human intervention.

The environment records examples at 90 frames per second, which is a standard frame rate for VR. Each example includes the position and orientation of head and hands, the label of the gesture being performed at that frame and its progress value (or *No gesture* for the inter-gestural segments). The progress value is also automatically estimated by the environment; given the initial and final frame of each gesture, this value is simply linearly interpolated between zero and one throughout the duration of the gesture.

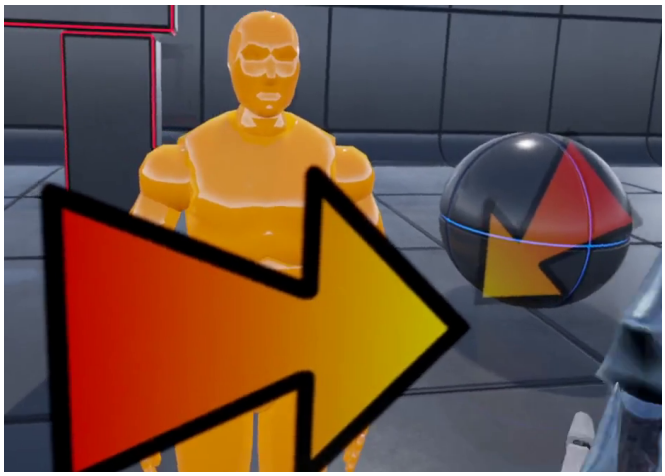


Figure 5-2: Gesture data collection environment. Arrows indicate the direction of the strike that the data collection participant must perform. The stationary character in orange serves as a reference for the gesture.

5.4.3 Data Preprocessing

The motion data registered by the VR hardware contains all the necessary information for gesture detection, but it cannot be directly used as provided. One common problem with current VR technology is that, in the general case, the system does not have any measure of where the body of the player is located beyond the three tracked points. As far as the VR hardware can tell, there is no difference, for example, between rotating the whole body in one direction or just turning the head and the hands in the same direction. More generally, the same reading from the VR tracking sensors may correspond to different body poses, and this is simply a limitation of the technology. This means there is no known coordinate system centred on the body of the player, and so the available spatial information is always referred to the centre of the scene instead. But this is not suitable to do gesture recognition; the detection of a gesture should not depend on the location or orientation where it is performed, so the collected data needs to be reframed in a coordinate system that is relative to the current position of the player.

To this end, we compute an estimate for the player's body location based on the position of the head. We assume that the player will be looking forward most of the time, but in some occasions they may turn the head to look briefly to one side, in what can be a very quick movement. Simply using the head as a reference for the body can therefore be unstable. But we can take the head pose and smooth its motion through time, applying a low-pass filter that mitigates the impact of those "high frequency" changes. A simple

method to do this is to use what is known as an exponential smoothing rule. Given the head pose at some frame t, with location $\boldsymbol{p}_H^t \in \mathbb{R}^3$ and rotation $\boldsymbol{r}_H^t \in SO(3)$, we can compute their corresponding filtered versions $\boldsymbol{p}_F^t$ and $\boldsymbol{r}_F^t$ as:

$$\begin{aligned} \boldsymbol{p}_F^0 &= \mathbf{0}_P \\ \boldsymbol{r}_F^0 &= \mathbf{0}_R \\ \boldsymbol{p}_F^t &= \mathit{Lerp}(\boldsymbol{p}_F^{t-1}, \boldsymbol{p}_H^t, \alpha_P) \quad t > 0 \\ \boldsymbol{r}_F^t &= \mathit{Slerp}(\boldsymbol{r}_F^{t-1}, \boldsymbol{r}_H^t, \alpha_R) \quad t > 0 \end{aligned} \tag{5.1}$$

Where $\mathbf{0}_P$ and $\mathbf{0}_R$ are respectively the null location and identity rotation, $\mathit{Lerp}\colon \mathbb{R}^3 \times \mathbb{R}^3 \times \mathbb{R} \to \mathbb{R}^3$ is the 3D linear interpolation function, $\mathit{Slerp}\colon SO(3) \times SO(3) \times \mathbb{R} \to SO(3)$ is the spherical interpolation function that interpolates rotations with uniform angular velocity around a fixed axis (Shoemake, 1985) and $\alpha_P \in [0, 1]$ and $\alpha_R \in [0, 1]$ are the location and rotation smoothing factors. These parameters determine how much the filtered estimation "lags" behind the head pose, in terms of location and rotation. They can be tuned from zero, in which case the body pose would never move from the initial null estimation (which would be equivalent to taking the head and hand positions with respect to the centre of the scene in absolute terms), to one, where the body pose estimation would always match the head.

Using the filtered head position as reference point for the control input makes it feasible to learn consistent gestural patterns from the data. Nonetheless, it is still important to have a sufficient number of examples to create a general model that reliably identifies the gestures of different players with different sizes and styles. The gesture recognition model cannot learn gestures that are not part of the dataset, so ideally we would want to capture the motion of every possible kind of player. This is hardly practicable, though, and it also makes it very costly to add a new gesture to the vocabulary as part of an iterative design process. However, it is possible to improve the variety and richness of a dataset using synthetic data augmentation (Wong et al., 2016). This technique consists of producing new examples derived from the already existing ones, applying different kinds of perturbations or distortions to them. The idea, in the context of gesture recognition, is that a slightly different execution of a gesture should still be classified with the same label, so making minor alterations in the control data should give equally valid training examples. This improves the robustness of the gesture recognition module, as it reduces the chances of overfitting to the training data, reinforcing the fundamental patterns behind each gesture and thus making it less sensitive to noise. For each recorded sequence of gestures, we consider the following four simple variations:

- A random scale factor $k_S \sim \mathcal{U}(0.85, 1.15)$ applied to the tracked control positions in the gesture, representing different body sizes and proportions.
- A random low-frequency sinusoidal perturbation with amplitude $A_P \sim \mathcal{U}(0, 10)$ and frequency $f_P \sim \mathcal{U}(0, 1)$ applied to the trajectories of the hand positions, representing variations in the execution of the gesture.
- A random low-frequency sinusoidal perturbation with amplitude $A_R \sim \mathcal{U}(0, \pi/12)$ and frequency $f_R \sim \mathcal{U}(0, 1)$ applied to the orientation of head and hands, representing differences in the articulation of the body during the gesture.
- A random time scaling factor $k_T \sim \mathcal{U}(0.85, 1.15)$, representing different speeds of gesture execution.

The magnitude of these operations is generally reasonable for most cases; for example, the scaling operation would transform the proportions of an average 170.0 cm tall person to those of a person between 144.5 cm and 195.5 cm tall. However, particular cases could call for a different configuration of these operators.

The combined application of all of these perturbations to a given gesture results in a significantly different variation that is still recognisable as the same gesture. Using this method, the available dataset can be arbitrarily augmented to any size.

Once the final dataset of examples is produced, the last data preparation step is defining the features that will be used by the gesture recognition model. While it would be possible to construct a model based on the controller positions alone, an adequate selection of additional features can facilitate the training process and improve the accuracy of the detection. Let $(\boldsymbol{p}_H^t, \boldsymbol{r}_H^t)$, $(\boldsymbol{p}_L^t, \boldsymbol{r}_L^t)$ and $(\boldsymbol{p}_R^t, \boldsymbol{r}_R^t)$ be the locations and rotations of the head, left hand and right hand at some frame t with respect to the estimated body pose $(\boldsymbol{p}_F^t, \boldsymbol{r}_F^t)$ (which can therefore be considered as the origin of coordinates at frame t). The concrete set of features for the model can be adjusted to the particular scenario at hand. In the case of sword fighting, not all of these are equally relevant. The head information (once it has been used to estimate the body pose) would only be useful with gestures including head actions, such as shaking the head, which are not part of the defined vocabulary. On the other hand, since we are considering two-handed sword fighting, both hands provide essentially the same information. It is therefore reasonable to use the position of one of the hands, $(\boldsymbol{p}_R^t, \boldsymbol{r}_R^t)$, as the basis for gesture recognition in this case. From this, additional simple features can be computed to provide the recognition model with useful information. Firstly, using the information from previous frames, it is possible to estimate the hand linear velocity $v_R^t \in \mathbb{R}$ and acceleration $a_R^t \in \mathbb{R}$:

$$
\begin{aligned}
v_R^t &= \frac{\|\boldsymbol{p}_R^t - \boldsymbol{p}_R^{t-1}\|}{\delta} \\
a_R^t &= \frac{\|\boldsymbol{p}_R^t - 2\boldsymbol{p}_R^{t-1} + \boldsymbol{p}_R^{t-2}\|}{\delta^2}
\end{aligned}
\tag{5.2}
$$

Where $\delta = 1/90$ is the duration of each captured frame.

Secondly, expressing the position of the hand in spherical coordinates (see appendix B) with origin at $(\boldsymbol{p}_F^t, \boldsymbol{r}_F^t)$ also gives us additional relevant features. These include the the radius or distance from the hand to the origin $\rho_R^t \in \mathbb{R}$, the inclination angle with respect to the vertical direction $\theta_R^t \in \mathbb{R}$ and the azimuth with respect to the horizontal plane $\phi_R^t \in \mathbb{R}$. These can be useful in the identification of gestures, but they would be difficult to infer by the model from the 3D coordinates, since they have a non-linear correspondence with these.

The complete input vector to the model is then $\boldsymbol{x}^t = \{\boldsymbol{p}_R^t, \boldsymbol{r}_R^t, v_R^t, a_R^t, \rho_R^t, \theta_R^t, \phi_R^t\} \in \mathbb{R}^{12}$, where the rotation $\boldsymbol{r}_R^t$ is expressed as a quaternion (see appendix B). The label for the example is made of two objects. On the one hand, each captured frame is labelled with a class $c^t \in \mathcal{C}$ where $\mathcal{C}$ is the set of all gestures under consideration plus a special class *No gesture* for non-gestural frames. This is encoded as a one-hot vector $\boldsymbol{y}^t \in \{0,1\}^{|\mathcal{C}|}$ defined as:

$$
y_i^t = \begin{cases} 1 & \text{if } Id(c^t) = i \\ 0 & \text{otherwise} \end{cases}
\tag{5.3}
$$

Where $Id \colon \mathcal{C} \to \{1, \ldots, |\mathcal{C}|\}$ is a fixed bijection that assigns an integer index to each gesture. The second element of the label is the gesture progress label $Q^t \in [0,1]$, directly provided by the data capture process. The full label for the example $\boldsymbol{x}^t$ is then $(\boldsymbol{y}^t, Q^t)$.

5.4.4 Gesture Model

The core of the gesture recognition module is the data-driven model responsible for identifying the gestures. As described, its mission is to provide a gesture recognition label on each frame according to the perceived user input. Conventional approaches to data-driven gesture recognition, such as hidden Markov models (discussed in chapter 2), generally fall short in one or other regard for our purposes. They are not well suited for real-time gesture detection, and do not provide a straightforward method to predict

the progress of the gesture. A neural network is a good fit for this purpose, as it is a powerful classification tool that can easily be designed to be fast enough for real-time operation. It also offers the flexibility to choose the network architecture that best suits the needs of the problem. The simplest option would be to take $\boldsymbol{x}^t$ as input and train the model to predict $(\boldsymbol{y}^t, Q^t)$. This, however, is very unlikely to yield good results; it is not difficult to conceive particular input vectors that could be part of different gestures in different contexts, and for which there would be no consistently good classification. For example, a single frame of a *Right* strike and a *Left* strike can be virtually identical when taken without further context, in spite of being opposite gestures. Moreover, it is obvious that having a single frame of data as input cannot be enough to predict the progress value of the gesture.

As an alternative, we can concatenate a window of recent frames and use that as input to the model, instead of only looking at the latest one. This should give us the necessary context to make the right predictions, but there is a trade-off to it. The larger the window, the more context, but also the more parameters to the neural network. This means that taking many frames as input would make the model more complex, slower and harder to train. Also, treating the window of frames as a big, unstructured vector does not take advantage of its underlying structure, making the recognition of patterns more difficult than necessary. These issues can be addressed with an architecture inspired by the Wavenet model (Oord et al., 2016), applied on a small scale. This model uses a stack of dilated convolutions (Yu and Koltun, 2015) in order to make use of a large window of input frames with a relatively small number of parameters.

The main idea is shown in figure fig. 5-3. All layers in the network perform a one-dimensional convolution operation with a fixed kernel, which in the figure is three. This convolution uses a progressively larger dilation factor in each layer, growing exponentially. So, in the example, the first hidden layer has a dilation factor of one (no dilation, which is a conventional convolution), the second hidden layer a factor of three and the output layer a factor of nine. Increasing the dilation factor in this manner ensures that all frames in the input window take part in the model prediction while minimising the number of parameters on each layer.

To formalise this idea, let $\boldsymbol{x}^t \in \mathbb{R}^{D_{In}}$ be the input vector received at frame t, when the network predict the corresponding label $(\boldsymbol{y}^t, Q^t) = \boldsymbol{z}^t \in \mathbb{R}^{D_{Out}}$. Choosing a number $H \in \mathbb{N}$ of hidden layers with sizes $\{D_1, \ldots, D_H\}$, and an integer base dilation factor K (in the figure three). The output of the model $\widehat{\boldsymbol{z}}^t$ is then given by the following expression:

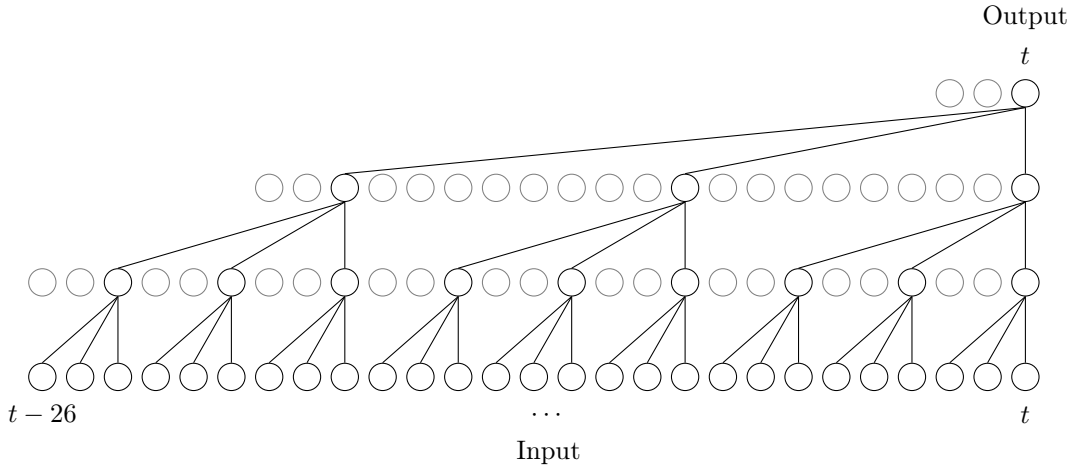


Figure 5-3: Dilated convolutions stack model with two hidden layers and base dilation factor of three. Only connections relevant to output at time t are shown.

$$\begin{aligned}
\widehat{\boldsymbol{z}}^t &= \widehat{\boldsymbol{z}}^t_{H+1} \\
\widehat{\boldsymbol{z}}^t_0 &= \boldsymbol{x}^t \\
\widehat{\boldsymbol{z}}^t_{i+1} &= A_{i+1}\left(Conv\left(\left[\widehat{\boldsymbol{z}}_i^{t-(K-1)K^i}, \ldots, \widehat{\boldsymbol{z}}_i^{t-(K-K)K^i}\right]^\intercal, W_{i+1}\right) + \boldsymbol{b}_{i+1}\right) \quad 0 < i \leq H+1
\end{aligned} \tag{5.4}$$

Where A_i is the activation function for the i-th layer and $W_i \in \mathbb{R}^{K \times D_i \times D_{i+1}}$ and $\boldsymbol{b}_{i+1} \in \mathbb{R}^{D_{i+1}}$ are its trainable convolution kernel and bias vector respectively. The function $Conv\colon \mathbb{R}^{K \times D} \times \mathbb{R}^{K \times D \times D'} \to \mathbb{R}^{D'}$ is the one-dimensional convolutional operator defined as follows:

$$Conv(X, W)_j = \sum_{k=1}^{K} \sum_{i=1}^{D} X_{ki} W_{kij} \tag{5.5}$$

What eq. (5.4) expresses is that the output of layer $i+1$ at instant t is computed from K outputs from the previous layer separated by a distance of K^i frames (the dilation factor of the layer) between each other. This maintains a constant convolution kernel size across the layers while maximising the overall receptive field, so in fact $\widehat{\boldsymbol{z}}^t_{i+1}$ is influenced by all the latest K^{i+1} input vectors, and in particular the model output $\widehat{\boldsymbol{z}}^t$ uses a total of K^{H+1} frames of input data. The resulting architecture is therefore capable of providing the necessary context for gesture detection while maintaining a

low overall complexity, making it a good choice for our framework.

5.4.5 Experimental Setup

Having defined the data acquisition workflow and model architecture, we may now construct the gesture recognition module for our sword fighting scenario. Using the methodology presented earlier, we collected a dataset of approximately twelve minutes of data recorded at 90 frames per second (over 64 000 frames). The data was preprocessed as described above, estimating the body position according to eq. (5.1) with smoothing factors $\alpha_P = 0.9$ and $\alpha_R = 0.9$. The examples were randomly split in 80% for training and 20% for evaluation, uniformly sampled from all gestures. The training split was augmented as discussed to produce up to six new examples per frame of data, for a total of approximately 365 000 training examples.

The selected dilated convolution model was set to use a base dilation factor of $K = 4$ with $H = 2$ hidden layers. This means each prediction of the model is computed using the latest $4^{2+1} = 64$ frames of input, or approximately 700 ms, which is longer than the total duration of over 95% of the gestures in the dataset. Each hidden layer has 100 convolutional channels at the output and use the hyperbolic tangent as activation function. The output of the models is structured in a slightly peculiar way. For the gesture classification, instead of simply having a softmax output with the nine possible classes (the eight kinds of strikes gestures plus the *No gesture* class), as would be the standard in a multi-class classification model, it contains eight independent sigmoid units corresponding to the eight actual gestures, producing a classification vector $\boldsymbol{c}^t \in [0, 1]^8$ for each input vector $\boldsymbol{x}^t$. This is used to compute an additional classification value for the *No gesture* class, defined simply as $1 - \max(\boldsymbol{c}^t)$, which completes the class prediction vector $\widehat{\boldsymbol{y}}^t \in [0, 1]^9$. From there, the class corresponding to the highest value in $\widehat{\boldsymbol{y}}^t$ is picked as the predicted class. What this means is that the *No gesture* class, instead of being trained as a proper type of gesture, will only be picked when no other class yields a prediction value over 0.5. The goal of this scheme is to prevent the model from attempting to learn what is a *No gesture*, since it is not really any particular action, and instead model it as anything that does not fall into any other class. This way, if a player adopts an idling position that is different from those in the dataset, it will be less likely to be mistaken for one of the gestures in the vocabulary. The prediction of the progress value, on the other hand, is given by a single sigmoid output unit $\widehat{Q}^t \in [0, 1]$ in the last layer, parallel to the class prediction.

Since the model is performing two tasks at the same time, namely gesture classification and progress prediction, the optimisation goal used to train it must be a combination

of two loss values corresponding to each of them. For the classification task, the most common option would be the softmax cross-entropy of the predictions and the one-hot encoded class. However, since each class is represented as an independent sigmoid value (as opposed to the result of a softmax operation), the sigmoid cross-entropy of the class label and the prediction is the most adequate choice, as it regards the prediction for each class as an independent binary classification problem. Progress prediction, on the other hand, is a regression task (even if the output values are also between zero and one). As we are interested in a continuous prediction instead of a binary one, the squared difference between the prediction and the label is better suited. Therefore, given an example $\boldsymbol{x}^t$ with label $(\boldsymbol{y}^t, Q^t)$ and a model prediction $(\widehat{\boldsymbol{y}}^t, \widehat{Q}^t)$, we define the objective loss function $L_G \colon \{0,1\}^9 \times [0,1] \times [0,1]^9 \times [0,1] \to \mathbb{R}$ as:

$$\begin{aligned} L_G(\boldsymbol{y}^t, Q^t, \widehat{\boldsymbol{y}}^t, \widehat{Q}^t) = &- \left(\sum_{i=1}^{9} y_i \log(\widehat{y}_i^t) + (1 - y_i^t) \log(1 - \widehat{y}_i^t) \right) \\ &+ G(\boldsymbol{y}^t) \eta_Q (Q^t - \widehat{Q}^t)^2 \end{aligned} \tag{5.6}$$

Where $G(\boldsymbol{y}^t)$ takes the value one if $\boldsymbol{y}^t$ encodes an actual gesture and zero if it encodes the *No gesture* class (in which case the progress value is irrelevant), and η_Q is a constant term that balances the relative importance of the classification and progress prediction tasks, experimentally fixed to $\eta_Q = 50$.

In addition to this, an example weighting scheme is also used. As the *No gesture* class is overwhelmingly more prevalent in the data than any other class (since it appears between every pair of gestures, for a period that is frequently no shorter than an actual gesture), the model is at risk of becoming biased towards that class (or, equivalently, negatively biased towards every gesture class). The reason is that a model that misclassifies most gestures as *No gesture* may still yield a low loss value only because *No gesture* examples dominate the dataset. To mitigate this effect, a weight value is assigned to each example according to its labelled class in order to increase the importance of underrepresented classes. Given the set of all classes $\mathcal{C}$ (including *No gesture*), then the class weight $W \colon \mathcal{C} \to \mathbb{R}^+$ is given by:

$$W(c) = \frac{\sum_{c' \in \mathcal{C}} N(c')^\beta}{|\mathcal{C}| N(c)^\beta} \tag{5.7}$$

Where $N(c)$ is the number of examples of class c in the training data and the exponent

Class	Examples	Precision	Recall	Most correct (%)
No gesture	125	0.928	0.747	90.4
Up	23	0.688	0.819	100.0
Up-right	15	0.576	0.918	100.0
Right	12	0.676	0.939	100.0
Down-right	17	0.760	0.912	100.0
Down	13	0.676	0.833	100.0
Down-left	8	0.645	0.862	100.0
Left	16	0.831	0.899	100.0
Up-left	12	0.557	0.842	100.0

Table 5-1: Per-class classification accuracy results for the gesture recognition model. Examples are complete gestures (or continuous *No gesture* sequences), not individual frames. Last column shows the percentage of examples with most frames correctly classified.

$\beta \in \mathbb{R}^+$ regulates the relationship between class imbalance and weights. Using $\beta = 0$ would produce uniform weights, while $\beta = 1$ yields an inversely proportional relation between class prevalence and weight. For this experiment, this value was set to $\beta = 0.7$. The class weights are then used to compute a weighted average of the loss value of the examples in a batch. Let $B = \{(\boldsymbol{y}^1, Q^1, \widehat{\boldsymbol{y}}^1, \widehat{Q}^1), \ldots, (\boldsymbol{y}^M, Q^M, \widehat{\boldsymbol{y}}^M, \widehat{Q}^M)\}$ be the tuples of corresponding labels and model predictions for a batch of M examples. The computed loss for the batch is then given by the following expression:

$$\frac{\sum_{t=1}^{M} W(C(\boldsymbol{y}^t)) L_G(\boldsymbol{y}^t, Q^t, \widehat{\boldsymbol{y}}^t, \widehat{Q}^t)}{\sum_{t=1}^{M} W(C(\boldsymbol{y}^t))} \tag{5.8}$$

Where $C(\boldsymbol{y}^t) \in \mathcal{C}$ is the class encoded by the one-hot vector $\boldsymbol{y}^t$. As examples of less represented classes will receive higher weights, the corresponding gradients will also be scaled up and they will have an overall more significant effect in the training process.

The model is trained in this manner using Adam optimisation for 10 000 epochs, or complete passes through the complete training dataset, on batches containing 25 sequences of up to 2 000 frames per sequence. To regularise the model and reduce overfitting, a dropout rate of 0.5 is used during training (see appendix A).[1]

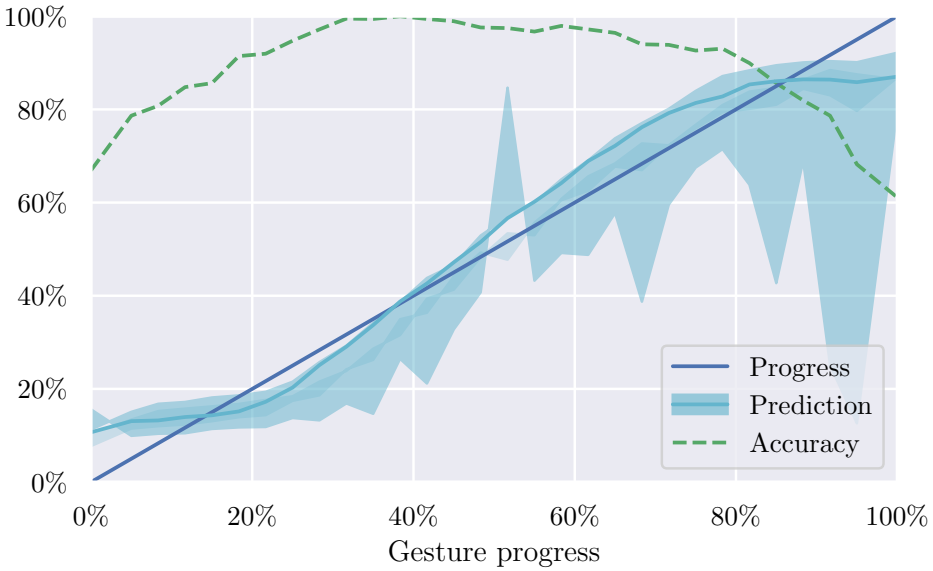


Figure 5-4: Gesture progress prediction results. The dark blue line shows the actual gesture progress, and the light blue line the average prediction at that point. The darker and lighter areas represent the range of the 50% and 95% best predictions respectively. Gesture classification accuracy is shown in green for reference.

5.4.6 Results

The factors to consider in the evaluation of the trained gesture recognition model are defined by the two tasks it performs. On the one hand, the model must accurately classify the player gestures, and on the other hand it must provide useful gesture progress predictions. In terms of classification accuracy, table 5-1 shows a breakdown of the model performance for each of the gestures, expressed through different metrics. The precision is the number of frames of a class correctly classified divided by the total number of predictions for that class (correct and incorrect). The recall is the number of frames of a class correctly classified divided by the number of frames of that class in the data. As can be seen, the model strongly favours a high recall over precision, except for the *No gesture* class. This means that the model will be biased towards finding gestures in the input at the expense of more false negatives. For example, a slight move of the hands by the player may be momentarily be identified as the beginning of a strike. This is however fine for our purposes, and it reflects a kind of mistake that an actual human sword fighter might make. It is also generally a short-lived error, as indicated by the last column of the table: all gesture examples are given at least a mostly correct classification, and only a minority of *No gesture* sequences are notably misclassified.

[1]Code and data available at `https://doi.org/10.15125/BATH-00752` (Dehesa, 2020).

The accuracy of the gesture progress prediction is represented by fig. 5-4. The results are not surprising: the progress prediction becomes close to the actual progress of the gesture in the middle region of a gesture, and deviates more from the correct value at the beginning and end. This is a consequence of the fact that the boundaries of a gesture are not precisely defined. In practical terms, the exact moments at which the controller button is pressed and released during the data collection phase varies with each gesture execution, so the progress value at both ends of the gesture becomes less certain. The variation in the length and speed of each gesture also contribute to this behaviour. Nonetheless, as the next section discusses, the behaviour of the character will only be conditioned by whether the gesture is in its first half (starting) or its second half (ending), which this model can reliably determine.

5.5 Behaviour Planning

The behaviour planning module is the point in the system at which interaction designers can define the rules that direct the behaviour of the character. The key aspect here is that, unlike the other two modules, this one is fully human-driven, and its dynamics are transparent and predictable (excepting those that are explicitly random). A simple model for this behaviour that is already popular in video games design, and suits our needs well, is a state machine. This is an effective tool because it is simple to use, which makes it accessible even for non-technical profiles, but also expressive, allowing to integrate a variety of behaviours without sacrificing clarity and maintainability. It also fits well with the rest of our framework, as the different states naturally express the behaviour that will be requested from the animation synthesis module.

Another advantage of state machines is that they can easily be tuned through parameters that alter or regulate the transitions, such as the probability of a transition event being triggered. This makes it possible to quickly experiment and iterate on the behaviour of the character without changing the underlying design. Behaviour parameters will be instrumental to definition of our different evaluation conditions in section 5.7.

It is worth remembering that the behaviour planner does not contain anything more than a completely abstract description of the character behaviour. As this section will show, this description can be fairly simple to conceive and to understand. However, the planner does not, by itself, implement any of the behaviours that it describes. That is the nature of the semantic plane of the interaction framework, which is constructed on simple abstractions that are exchanged with the physical level. The interface between this module and the rest of the system is composed of the events that are detected

from the environment and the states that the character is requested to perform. Unlike a conventional system, in which one would have to program the specific algorithms to detect such events and perform such states, the data-driven modules of the system carry out these tasks using the collected data to guide their actions.

5.5.1 Sword Fighting Behaviour

In the context of our sword fighting scenario, there are, in essence, two types of action that the character may take: defend or attack. We could envision the behaviour of the character as a sort of event-driven system where the task of the planner would be to switch from one to another kind of action depending on the perceived events, which can be external or internal to the planner itself. For example, when a strike from the user is detected, the character may react adopting a defensive stance. This kind of logic is what the behaviour planner must encode in the form of a state machine.

We define the following four principles to guide the design of the character behaviour:

- Before engaging in any actual sword fighting, the character must first get sufficiently close to the position of the player.
- The character must act defensively by default, attempting to block all incoming strikes.
- At random intervals, the character may initiate an attack against the player
- The character may also attempt to counter-attack strikes from the player.

These requirements are encapsulated by the state machine shown in fig. 5-5. Here, beginning from the *Start* point, the character goes first into an initial *Walking* state. This is not, strictly speaking, a sword fighting state, as its only purpose is to bring the character close enough to the player to actually engage in an interaction, which is represented by the event *Is close*. This is where the actual sword fighting begins, with the behaviour alternating between *Defending* and *Attacking* states.

The default state in the diagram is *Defending*. The character can stick to this behaviour indefinitely, as it represents a passive, reactive attitude. While defending, the character will generally adopt a neutral defensive pose with the sword in front of the body. There is no direction to this state, the character will simply attempt to block all kinds of strikes, moving the sword as necessary when the player's sword comes close to its body. There are two events that can pull the character out of the *Defending* state: a *Random timer* event and a *Player attacked* event. The *Random timer* is triggered at random time intervals, and it represents the offensive will of the character. After having spent some time defending itself, the character may decide to issue an attack towards the player,

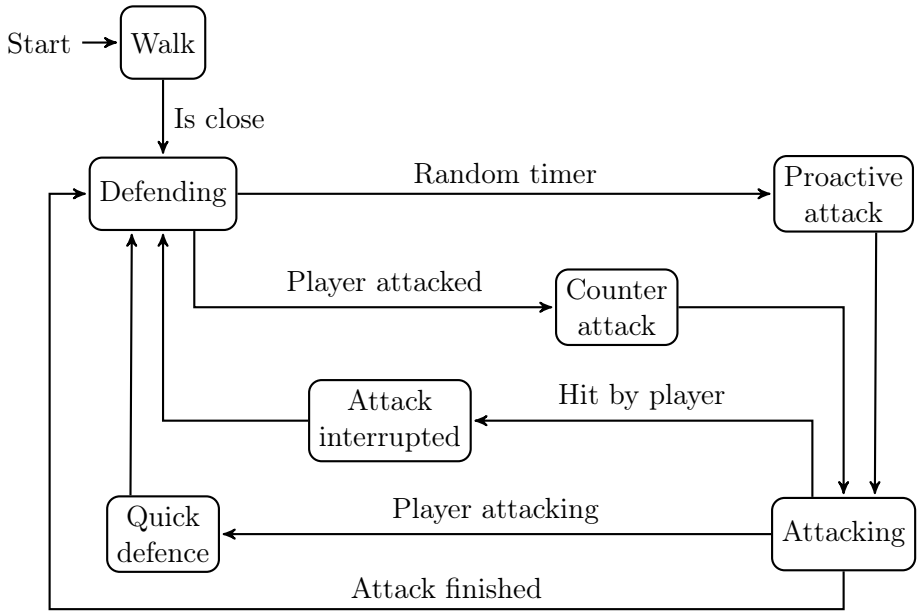


Figure 5-5: Character behaviour diagram

coming into the *Proactive attack* state. In this state, an strike direction is chosen at random and the attack is initiated, leading to the *Attacking* state. The other event that can motivate an attack is *Player attacked*, which represents the completion of an attack from the player. In this situation, the character may decide to start a *Counter attack*. This is similar to a *Proactive attack*, but in this case the strike direction is not picked at random. Instead, the character will attempt an attack in the direction that the player has left more unprotected during the attack. So, for example, if the player performed an attack from right to left, the right side will have been left more open to an attacker, and so the character would start a strike from left to right (from its own point of view). Likewise, a player strike from the top right to the bottom left would be countered with a strike from the top left to the bottom right, and so on. In short, every time the character counter attacks it will perform a symmetric strike, which takes us again to the *Attacking* state.

The character will usually stay in *Attacking* until the strike is completed, signalled by the event *Attack finished*. A completed strike in this context can be a fully realised strike that has impacted the player, but not necessarily. If the player blocks the incoming strike, the attack is equally finalised. In both cases, the character will come back to the *Defending* state. However, there are circumstances that may induce the character to interrupt its strike before completion. On the one hand, if a new incoming strike from the player is detected, the event *Player attacking* is issued, which may lead to a *Quick*

defence. This is a defensive reaction from the character that cancels its current attack in an attempt to protect itself from a potential hit by the player. On the other hand, it may happen that the character actually gets hit during the attack, represented by the event *Hit by player*. The hit simply stops the action from the character, transitioning from *Attacking* to *Attack interrupted*. In all cases, the *Attacking* state is always transitory and eventually always leads back to *Defending*.

5.5.2 Event Detection and Configuration

Events are what ultimately trigger changes in the behaviour of the character, and it is important to establish the correspondence between their semantic representation and their physical counterparts. Some of these events have a trivial practical implementation. *Start* is simply issued at the creation of the virtual environment. *Is close* happens as soon as the measured distance between the character and the player is small enough to begin the sword fighting. *Hit by player* represents every collision between the player's sword and the virtual character (or a simplified collision shape around it). And *Random timer* is a completely internal event that is generated at random intervals. These kind of events are extremely common in video games and other similar media, not necessarily in VR.

Other events are more complex and need to be informed by components of the interaction framework. The event *Attack finished* is a simple notification from the animation synthesis module to acknowledge the end of an attack animation. This event does not need to carry any additional information, as it invariably directs the character to the *Defending* state. The gesture recognition module provides two kinds of event: *Player attacking* and *Player attacked*. These are similar events that are issued as the system identifies strikes in the player input, and they indicate the specific direction of the incoming strike. The difference between the events is the time at which each one is generated. Using the gesture progress prediction provided by the gesture recognition module, one can define relevant points within the gesture execution. Since we are only interested in the beginning and the end of each gesture, we can simply take the midpoint of the gesture (a normalised progress value of 0.5) as the split point for the events. Thus, when a gesture is first recognised with a progress under 0.5, a corresponding *Player attacking* event happens. Later, after the gesture is recognised with a progress over 0.5, the respective *Player attacked* event is sent.

The interesting aspect of this exchange of events is that it allows for certain parameterisation of their relationship between them and their effect on the behaviour. This enables a designer to fine tune the characteristics of the character, or even to create a

varied range of characters with different behaviours using all the same components and design. The most obvious configuration parameter is the frequency at which *Random timer* is triggered. This largely determines the aggressiveness of the character, ranging from an incessant attacker to a careful defender. As a designer, a convenient way to think about this is in terms of the rate $\lambda_A \in \mathbb{R}$ at which attacks are issued, as a mean number of attacks per second from the *Defending* state. Using a negative exponential probability distribution with rate λ_A to model the intervals at which *Random timer* is generated achieves exactly this. The time until the next attack $t_A \in \mathbb{R}$ from the moment the character goes into the *Defending* state is expressed as:

$$t_A = -\frac{\log(1-r)}{\lambda_A} \tag{5.9}$$

Where $r \sim \mathcal{U}(0,1)$ is sampled on entering into the state. This a varied but consistent behaviour that will, on average, respect the λ_A rate.

In addition to this, other characteristics can also be parameterised to boost or discourage some reactions. In particular, there are two other actions that are proactively initiated by the character and can be subject to regulation, namely *Counter attack* and *Quick defence*. In both of these cases, the character tries to seize an opportunity to either hit the player or better protect itself, but they are not necessary reactions as, for example, the state *Attack interrupted* after being hit by the player's sword. Therefore, we can adjust the ability of the character to exploit these opportunities, representing also its defensive or offensive attitude, but also its "skill" as a sword fighter. This can be simply done with two corresponding probability values, $p_C \in \mathbb{R}$ and $p_D \in \mathbb{R}$, one for each of these state transitions. When the character is *Defending* and *Player attacked* happens, it will only transition to *Counter attack* with probability p_C. Likewise, if the character is attacking and *Player attacking* is detected, a *Quick defence* will only take place with probability p_D. These probabilities can be set to some intermediate values between zero and one, or exactly to zero or one, in order to completely enforce or suppress the behaviour. Different configurations would then reflect different kind of fighters, as, for example, a very aggressive character, which may try to counter attack on every occasion, but fail to turn to a quick defence while striking.

The parameters λ_A, p_C and p_D have a significant effect on the behaviour of the character, and demonstrate the flexibility of the framework to adapt to different needs and designs. In section 5.7 we will study the impact that different configurations have on the perception of the character from the perspective of the user.

5.6 Animation Synthesis

The animation synthesis component is the final element that completes the interaction framework. It is responsible for bringing the actions requested by the behaviour planner to life through convincing animation. It is, in a sense, the most critical piece of the system, since it produces the actual result displayed to the player. While poor gesture recognition or behaviour design may impact the perceived "intelligence" of the character, implausible animation would immediately break the believability of the interaction as a whole. Using a data-driven model for animation synthesis ensures that all the actions will be at least based off actual motion capture data.

We have already discussed data-driven animation in the context of quadruped locomotion in chapter 4. As before, the goal here is to automate the extraction of plausible motion from the available data without manual intervention from an animator. The methodology is thus similar to what has already been presented, but with important differences to incorporate the larger context of real-time interaction with the player.

5.6.1 Sword Fighting Animation

When considering the animation of a sword fighting character, we again recognise attacking and defending as the two main types of action to perform, and the essential differences between these two. Attacks are determined by the behaviour planner to take place at some specific time and with some specific direction (which may or may not have been picked at random), and they must be performed predictably according to the performance of a motion-captured actor. This is a simple task, as it does not require any further consideration. The moment an attack needs to be issued, a strike animation can be played to that effect. Defending, nonetheless, is a more difficult function to emulate. A defence from a strike is not an action that is decided in advance, but rather a reaction to an action from the player. And, since the trajectory of the strikes from the player are varied and difficult to predict, picking the best animation from an existing library to block the strike is not easy. This is the case where a data-driven model can be more effective than a complicated hand-crafted animation selection algorithm.

Therefore, it makes sense to take a hybrid approach to animation in the case of sword fighting. On the one hand, attack animations can be played out as fixed sequences of offensive motion capture data. Having a collection of attack motion clips for each of the strike directions, the instruction to perform an attack from the behaviour planner is reduced to picking a corresponding clip from this collection and replaying it. Meanwhile, defensive actions are synthesised using a machine learning model trained on defensive

motion capture data. Transitions between these two modes are realised as a simple linear blend between the animation synthesised by the model and the attack clip, and viceversa, over a short period of time (0.5 s), giving an overall continuous appearance to the animation.

The remaining of this section discusses the construction of the animation synthesis model for character defence.

5.6.2 Data Collection

The animation synthesis model requires an adequate dataset of motion capture clips to be trained. Unlike the case of gesture recognition, this must be collected using specialised motion capture equipment, since it is necessary to record the position of the whole body. The need for such specialised equipment may be a barrier in some circumstances, but motion capture technology is already commonly used in the digital entertainment industry, and in most cases it is already a prerequisite for highly-realistic animation production. While hand-authored animation clips could theoretically be used as a data source, it is unlikely to obtain organic results from them. The advantage of motion capture data is precisely the rich blend of irregularities and variations in the performances that are very difficult to recreate by hand.

Capturing animation data for interaction requires the participation of two actors, one acting as the character and another interpreting the player. There are two important reasons for this. First, it is generally difficult to convincingly perform interactive actions without the interacted party. For example, the motion to block a sword strike is unlikely to be realistic without an actual impact of two sword props. But, importantly, on a technical level, having a second actor makes it possible to also capture the performance corresponding to the player (or part of it). This is useful because it allows us to incorporate the control data from the player directly as an input to the animation model. This control data, as has already been described, consists of the tracked head and hand positions, and it is crucial to be able to synthesise adequate reactions. But using it requires knowing what was the player input for each clip of training data, which does not exist as such, since the motion data is not captured in a VR environment. The captured motion of the actor interpreting the player serves as a proxy to this control data, and establishes the link between the motion capture domain and the VR control domain.

In the particular case of a sword fighting scenario, the interaction is actually centred around the virtual swords wielded by the player and the character more than the body or

hands themselves. It is true that the position of the sword is mathematically computed from the position of the hands, but it is simpler to reason about sword positions than hand positions in this case. For this reason, the position of the tip of the swords is also included among the tracked points in the motion capture sessions. This will facilitate the construction of additional input features as part of the data preparation steps.

The captured motion data includes the offensive and defensive character actions. Each kind of action is captured separately, which makes it easy to, one the one hand, categorise the captured data and, on the other hand, ensure that a balanced number of strikes and blocks are captured in each direction. The attack animation data is split in individual strike clips classified by direction, which are used to animate the offensive actions of the character as described above. The sword blocking animation data is collected into a dataset with no further manual labelling or editing.

5.6.3 Data Preprocessing

The output of the motion capturing process is a database of frames containing the body pose of the actor playing the virtual character, represented as a skeletal model with $J = 25$ animated joints, and the position of both swords involved in the interaction. The captured motion data for each frame t is summarised in a vector $\boldsymbol{m}^t = \{\boldsymbol{p}_1^t, \ldots, \boldsymbol{p}_J^t, \boldsymbol{r}_1^t, \ldots, \boldsymbol{r}_J^t, \boldsymbol{h}_1^t, \boldsymbol{s}_1^t, \boldsymbol{h}_2^t, \boldsymbol{h}_2^t\}$. Each $\boldsymbol{p}_i^t \in \mathbb{R}^3$ represents a spatial location of the i-th joint with respect to the root of the character (the ground projection of its centre of gravity), and each $\boldsymbol{r}_i^t \in \mathbb{R}^6$ is its corresponding orientation, expressed as the rotated unit vectors along the x and y axes (see appendix B). The pairs $\boldsymbol{h}_1^t$ and $\boldsymbol{h}_2^t$ are the locations of the sword hilts of the character and the player respectively, and $\boldsymbol{s}_1^t$ and $\boldsymbol{s}_2^t$ are the location of the sword tips. Unlike the case of gesture recognition, here we will not be considering using data augmentation techniques, since those are not as useful for this kind of regression task as they are for classification problems. The reason is that the model is trained to emulate the patterns in the data, and random alterations, while still effective in making it less sensitive to outliers, would destroy details in those patterns. Hence, all the training data must be produced upfront through motion capture sessions.

In essence, the job of the animation synthesis model is to predict the pose at $t + 1$ with the information received up until t. But, instead of simply attempting to estimate $\boldsymbol{m}^{t+1}$ from $\boldsymbol{m}^t$, it is useful to derive additional features that provide more context to the model. In this case, as this is not a locomotion model, we are not interested in the trajectory of the character, which will always stay in the same place. However, the trajectory of the sword tips is actually a very important piece of information to take

into account. In particular, knowing the 3D speed and acceleration of each sword would allow the model to identify whether the player's sword is getting closer or further away, which should obviously produce different reactions. Providing three recent positions of the sword tips instead of just the most recent makes this possible for the model. Since the positions of the sword tips can be slightly unstable, instead of just using frames $t-2$, $t-1$ and t we pick a slightly larger step size $S = 3$ and use the sword tip positions from $t-2S$, $t-S$ and t.

In addition to that, some additional geometrical features are added too. First, the points $\boldsymbol{c}_1^t \in \mathbb{R}^3$ and $\boldsymbol{c}_2^t \in \mathbb{R}^3$, defined as the pair of closes points in the 3D segments defined by the swords, $\overline{\boldsymbol{h}_1^t \boldsymbol{s}_1^t}$ and $\overline{\boldsymbol{h}_2^t \boldsymbol{s}_2^t}$, as well as the distance between them $d^t = \|\boldsymbol{c}_1^t - \boldsymbol{c}_2^t\|$. These are helpful towards the goal of bringing the sword of the character close to the player's, which is how ultimately the blocking is done. Second, the angular positions of the player's sword with respect to the character. This is, for both the hilt and the tip of the sword, the inclination and azimuthal angles, $\boldsymbol{a}_2^t = \{\theta_{h_2}^t, \phi_{h_2}^t, \theta_{s_2}^t, \phi_{s_2}^t\} \in \mathbb{R}^4$, from the point of view of the character. These angles are strongly related to the direction of the incoming strike from the player.

The complete input vector to the model for frame t is constructed as $\boldsymbol{x}^t = \{\boldsymbol{s}_1^{t-2S}, \boldsymbol{s}_1^{t-S}, \boldsymbol{s}_1^t, \boldsymbol{q}_1^{t-2S}, \boldsymbol{q}_1^{t-S}, \boldsymbol{q}_1^t, \boldsymbol{s}_2^{t-2S}, \boldsymbol{s}_2^{t-S}, \boldsymbol{s}_2^t, \boldsymbol{q}_2^{t-2S}, \boldsymbol{q}_2^{t-S}, \boldsymbol{q}_2^t, \boldsymbol{c}_1^t, \boldsymbol{c}_2^t, d^t, \boldsymbol{a}_2^t, \boldsymbol{p}_1^t, \ldots, \boldsymbol{p}_J^t, \boldsymbol{v}_1^t, \ldots, \boldsymbol{v}_J^t\} \in \mathbb{R}^{D_{In}}$, where each $\boldsymbol{q}_i^t \in \mathbb{R}^3$ is a unit vector in the direction of $\overrightarrow{\boldsymbol{s}_i^t \boldsymbol{h}_i^t}$ and each $\boldsymbol{v}_i^t = \boldsymbol{p}_i^t - \boldsymbol{p}_i^{t-1}$ simply expresses the velocity of each joint. The total dimensionality of the input is then $D_{In} = 197$.

The corresponding output vector for that frame is defined as $\boldsymbol{y}^t = \{\boldsymbol{p}_1^{t+1}, \ldots, \boldsymbol{p}_J^{t+1}, \boldsymbol{v}_1^{t+1}, \ldots, \boldsymbol{v}_J^{t+1}, \boldsymbol{r}_1^{t+1}, \ldots, \boldsymbol{r}_J^{t+1}\} \in \mathbb{R}^{D_{Out}}$, with $D_{Out} = 300$, from which the next pose is reconstructed. The output of the model is then used to compute the input vector for the next frame, completing the autoregressive animation synthesis loop.

5.6.4 Animation Synthesis Model

Having a dataset of defensive captured motion, we can now build the model that will synthesise the blocking animation during the interaction. The main goal of this model is to "track" the sword of the player to be able to put the sword of the character in the right place at all times. Since that is the main conditioning factor of the desired result, a GFNN model, as introduced in chapter 3, is a good fit for this problem. Instead of having a regular neural network capture all the necessary patterns to learn to block any kind of incoming strike, a GFNN can subdivide the space of the problem to learn local patterns in a blended grid of smaller networks. This means that different parameters of

the model will specialise in different aspects, such as blocking strikes from the bottom left or the top right.

The key aspect in the design of a GFNN model is the definition of the grid dimensions. In this case, if we want to break down the problem in function of the incoming strike direction, then taking the horizontal and vertical position of the player's sword with respect to the character as grid dimensions should have precisely that effect. However, there is another important dimension to consider, which is the perpendicular distance between the character's body and the player's sword. In general, the character only needs to protect itself with the sword when the incoming strike is close, and can maintain a more natural, neutral defensive stance when the player's sword is at a distance. In sum, we shall have a 3D GFNN model in which each grid parameter vector $\boldsymbol{\gamma}^t \in [0,1]^3$ is directly derived from the 3D position of the player's sword. To do this, we define a static spatial region of interaction for the sword around the character. This region is a box with origin at $\boldsymbol{b} \in \mathbb{R}^3$ and dimensions $\boldsymbol{h} \in \mathbb{R}^3$, and it represents the boundaries containing the typical range of motion of the player's sword. We can then define $\boldsymbol{\gamma}^t$ as the normalised coordinates of the sword tip $\boldsymbol{s}_2^t$:

$$\gamma_i^t = \min\left(\max\left(\frac{s_{2i}^t - b_i}{h_i}, 0\right), 1\right) \tag{5.10}$$

Using these grid parameters allows us to associate each of the experts in the GFNN model with specific locations in space. Figure 5-6 visualises this structure. In grey is represented the animated character, as well as the sword of the player. The experts, represented in orange, specialise for each of the cases where the player's sword is up or down, left or right and near or far, making the task of synthesising new defensive poses much simpler for the network. On each frame, the position of the sword tip marked in green determines how these experts are blended according to the spline interpolation scheme from the model, so the behaviour of the character will smoothly navigate through the 3D grid space to produce reactions adapted to the input from the player.

5.6.5 Experimental Setup

We collected approximately 15 minutes of motion capture data recorded at 30 frames per second (over 26 000 frames) featuring defensive sword fighting actions to train the animation synthesis model. This data is split into 80% for training and 20% for evaluation purposes. The GFNN model is structured as shown in fig. 5-6, using a 3D region of interaction of 120 cm wide, 200 cm tall and 70 cm deep. The grid itself has four horizontal subdivisions, four vertical subdivisions and two depthwise subdivisions, for a

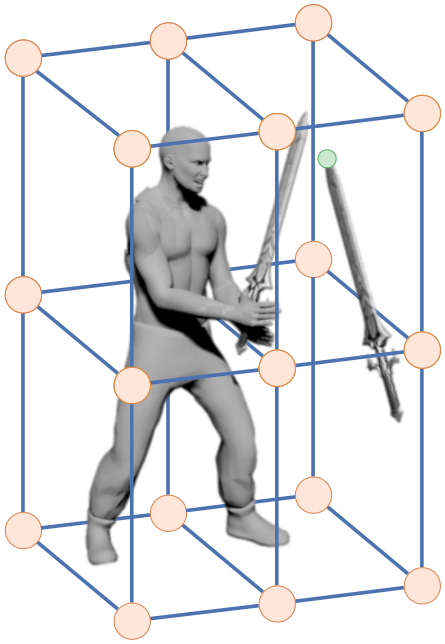


Figure 5-6: GFNN model structure for sword fighting animation synthesis.

total of 32 grid parameterisations. Each these contain the parameters to a base network architecture with two hidden layers of 55 units with ReLU activation.

Both input and output features are normalised for training by the mean and standard deviation of each component in the training data. For each training example $(\boldsymbol{x}^t, \boldsymbol{y}^t)$ with a prediction $\widehat{\boldsymbol{y}}^t$, the animation model optimisation goal $L_A \colon D_{Out} \times D_{Out} \to \mathbb{R}$ is simply the sum of squared differences between the prediction and the target:

$$L_A(\boldsymbol{y}^t, \widehat{\boldsymbol{y}}) = \left\| \boldsymbol{y}^t - \widehat{\boldsymbol{y}}^t \right\|^2 \tag{5.11}$$

The model is trained for 300 000 steps on batches of 32 examples, using a dropout rate of 0.3, an ℓ^2 regularisation factor of 100 and Adam optimisation with a learning rate of 0.0001 (see appendix A).[2]

5.6.6 Results

The quality of the animation synthesis model can be measured by comparing actual clips of motion data with the pose predictions yielded by the model for the same input. Starting from a common initial pose, and using the captured player's sword position as

[2]Code and data available at `https://doi.org/10.15125/BATH-00752` (Dehesa, 2020).

input, the animation synthesis model can produce a stream of predictions that should closely match the animation in the motion clip. However, measuring exactly how close these two animations are is not completely straightforward. The simplest approach would be to just aggregate the differences between the predicted and actual locations of each joint in every frame. This is a reasonable metric, but it does not account for the fact that some joints have a greater impact on the overall character pose than others. An error of a few centimetres in a joint would have a greater impact in the pose if it affects, say, and arm than a finger. For this reason, we instead design a "pose difference" metric that attempts to represent the error that the displacement of each bone introduces in the pose. Let $S = \{(i, j) \mid \text{joint } i \text{ is the parent of joint } j\}$ be the hierarchical structure of the skeleton driving the character. For each bone, a skew quadrilateral is defined by its actual position in the data and the model prediction for that bone. An estimation of the surface of this skew quadrilateral can be taken as the bone error. Given the joint positions at frame t, $P^t = \{\boldsymbol{p}_1^t, \ldots, \boldsymbol{p}_J^t\}$, and the corresponding joint positions predicted by the model $\widehat{P}^t = \{\widehat{\boldsymbol{p}}_1^t, \ldots, \widehat{\boldsymbol{p}}_J^t\}$, the pose difference function $D\colon \mathbb{R}^{3\times J} \times \mathbb{R}^{3\times J} \to \mathbb{R}$ is defined as:

$$D(P^t, \widehat{P}^t) = \sum_{(i,j)\in S} \left(\triangle(\boldsymbol{p}_i^t, \boldsymbol{p}_j^t, \boldsymbol{m}_{ij}^t) + \triangle(\widehat{\boldsymbol{p}}_i^t, \widehat{\boldsymbol{p}}_j^t, \boldsymbol{m}_{ij}^t) + \triangle(\boldsymbol{p}_i^t, \widehat{\boldsymbol{p}}_i^t, \boldsymbol{m}_{ij}^t) + \triangle(\boldsymbol{p}_j^t, \widehat{\boldsymbol{p}}_j^t, \boldsymbol{m}_{ij}^t) \right) \tag{5.12}$$

Where $\boldsymbol{m}_{ij}^t = \frac{1}{4}(\boldsymbol{p}_i^t + \boldsymbol{p}_j^t + \widehat{\boldsymbol{p}}_i^t + \widehat{\boldsymbol{p}}_j^t)$ and $\triangle\colon \mathbb{R}^3 \times \mathbb{R}^3 \times \mathbb{R}^3 \to \mathbb{R}$ is the triangle area function. The metric D provides a surface value that can be related to the dimensions of the character bones. Figure 5-7 shows a histogram of the resulting pose difference measured between the predictions of the animation synthesis model and the evaluation motion capture data. The results yield a mean difference of $0.427\,\mathrm{m}^2$, with a standard deviation of $0.267\,\mathrm{m}^2$ and a median of $0.361\,\mathrm{m}^2$. As shown in the figure, the majority of frames are predicted with a pose difference in the range between $0.25\,\mathrm{m}^2$ and $0.50\,\mathrm{m}^2$, a relatively small error with respect to the character human proportions. Table 5-2 shows a breakdown of the pose difference for the most important bones in the skeleton. Unsurprisingly, the single most significant bone is the one driving the sword tip position. It is worth noting, however, that an error in the position of the sword does not necessarily correspond to a bad defensive pose. As long as the sword is within the range of positions where it does effectively block (or attempt to block) incoming attacks, the behaviour can be considered acceptable. The next most significant bones are in the wrists, which are closely related to the sword position, followed by a more even distribution of the error across the rest of bones.

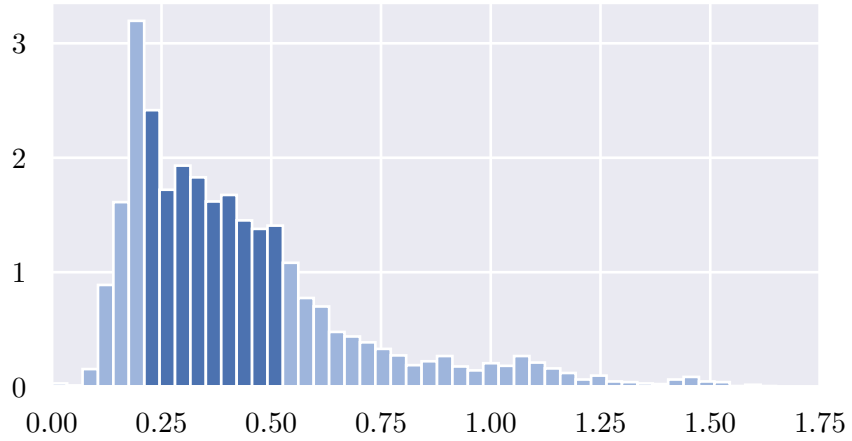


Figure 5-7: Histogram of the measured pose difference in square metres between the motion predicted by the animation model and the evaluation data. Highlighted area represents 50% of the examples around the median.

Bone	Avg. difference (m^2)	Error ratio (%)
Sword tip	0.120	28.1
Left wrist	0.036	8.5
Right wrist	0.033	7.6
Left elbow	0.023	5.4
Right elbow	0.022	5.2
(Rest of bones)	<0.021	<5.0

Table 5-2: Bones yielding the highest average pose difference in the evaluation data. Last column represents the ratio of the total error across all evaluation examples attributable to that bone. Last row figures are per bone.

5.7 User Evaluation

We have introduced all the elements of our VR interaction framework and described their application to a sword fighting scenario. The above sections present quantitative results on the data-driven models that have been trained, but we still need to analyse how the three components work together once they are integrated into a cohesive system. An interactive scenario like this cannot, however, be evaluated through simple performance and error metrics, as it is the quality of the overall experience that we are aiming to measure. The evaluation of the system needs then to be carried out through user studies that reflect the impressions that potential players would get.

User studies require the definition of comparable evaluation conditions in order to determine the impact of the evaluated factors. We designed the following three conditions:

Figure 5-8: Snapshots of the sword fighting evaluation scenario.

Control (C) The character is not animated by the interaction framework, but instead it directly uses motion capture clips and hand-crafted logic. In order to defend itself, the character repeatedly plays blocking animations when the player's sword approaches, chosen according to the position of the sword. These animations are interleaved with occasional clips of sword strikes.

Aggressive & Unskilled (A) A character with an attack rate of $\lambda_A = 1\,\mathrm{s}^{-1}$ and no ability to counterattack or quick-defend ($p_C = 0$, $p_D = 0$).

Defensive & Skilled (D) A character with an attack rate of $\lambda_A = 0.3\,\mathrm{s}^{-1}$ and very good ability to counterattack and quick-defend ($p_C = 1$, $p_D = 1$).

In the absence, to the best of our knowledge, of a comparable work to use as a baseline, the control condition C is a traditionally designed system using the same animation data with a simple programmed behaviour. Conditions A and D are on the one hand examples of characters animated through our framework, as compared to C, but also examples of different parameterisations of the behaviour described in section 5.5. This allows us to observe the effect of the different parameters on the user perception.

The evaluation is structured in two separate user studies.[3] A first questionnaire study was initially run with the goal acquiring broad feedback from a larger pool of participants. The second interactive study was run later with a more reduced set of participants to obtain a more accurate assessment of the perception that actual players might get from the system.

5.7.1 Questionnaire Study

In this study, participants were presented with one-minute videos of sword fighting sequences featuring the characters described by each of the conditions. The videos

[3]Data available at `https://doi.org/10.15125/BATH-00752` (Dehesa, 2020).

displayed the perspective of a VR player, with the opponent character walking towards the player and engaging in sword fighting, attacking and defending. Figure 5-8 shows snapshots of this point of view. After each of the videos, participants completed a questionnaire about their impressions, including questions from different scales. First, the Interest / Enjoyment scale from Intrinsic Motivation Inventory (IMI) (Ryan, 1982), a well-established questionnaire about the subjective experience of an activity that is frequently used in gaming research. The following questions were adapted from it:

- Watching the video was fun.
- I thought watching the video was boring.*
- The video did not hold my attention at all.*
- I would describe the video as very interesting.
- I thought watching the video was quite enjoyable.
- While I was watching the video, I was thinking about how much I enjoyed it.

Second, the following questions adapted from the Immersion questionnaire by Jennett et al., (2008), frequently used in VR user experience evaluation:

- I would like to play the game.
- Sword fighting in the video did not seem as real to me as it would be in the real world.*

For each of these, a score between one, meaning "Not at all true", to seven, meaning "Very true", was given. Questions marked with * had its score reversed.

The Immersion questionnaire items were grouped into a Realism subscale along with three more questions querying about the realism of the different actions of the opponent. These included "Walking", "Attacking" and "Defending", assessed in a seven-point scale from "Completely unrealistic" to "Completely realistic".

Finally, participants indicated their agreement with a set of adjectives applied to the behaviour of the character in the video. These adjectives included "Skilled", "Aggressive", "Intelligent", "Clumsy", "Repetitive", "Surprising" and "Rushed". Each of these was also scored in a seven-point scale from "Strongly disagree" to "Strongly agree".

A total of 41 participants took part in the study, aged from 18 to 63 (mean 31.7, s.d. 12.4). There were 11 females, 28 males and 2 participants with other or unspecified gender. Participants also completed a preliminary self-assessment of their experience in different relevant areas, on a scale from one ("No experience") to five ("Expert"). These included video games, with a mean score of 3.59 (s.d. 1.02); virtual reality, with

Scale	Cond. 1	Cond. 2	$t(40)$	p
IMI Interest	A	C	3.379	0.003^*
	D	C	6.405	$< 0.001^*$
Realism	A	C	4.400	$< 0.001^*$
	D	C	6.945	$< 0.001^*$
Skilled	A	C	5.792	$< 0.001^*$
	D	C	8.465	$< 0.001^*$
Aggressive	A	C	5.525	$< 0.001^*$
	D	C	5.234	$< 0.001^*$
Defensive	C	A	0.192	0.849
	D	C	1.368	0.358
Intelligent	A	C	3.936	$< 0.001^*$
	D	C	6.996	$< 0.001^*$
Clumsy	C	A	4.114	$< 0.001^*$
	C	D	5.163	$< 0.001^*$
Repetitive	C	A	1.840	0.073
	C	D	3.862	0.001^*
Surprising	A	C	1.865	0.070
	D	C	4.368	$< 0.001^*$
Rushed	A	C	1.521	0.272
	D	C	2.261	0.088

Table 5-3: One-tailed t-tests over the results of the questionnaire study. The hypothesis under test is that the measure for condition 1 is greater than that for condition 2. Starred rows indicate statistical significance ($p < 0.05$).

mean 2.24 (s.d. 1.09); video games development, with mean 2.22 (s.d. 1.39); 3D animation, with mean 1.78 (s.d. 0.85); and actual sword fighting, with mean 1.32 (s.d. 0.57).

Figures 5-9 to 5-11 (pages 108 to 110) show the results from the questionnaires in terms of the distribution of ratings across each of the seven-point scales. In general, the control condition C is perceived as inferior to A and D, giving place to lower IMI, Immersion and Realism ratings. Participants strongly associated this condition with adjectives like "Clumsy" and "Repetitive", while A and D were more frequently assessed as "Skilled" or "Intelligent". There are also interesting differences between A and D. While both exhibit comparable realism, condition D appears to attract more interest from the participants, and is seen as significantly more "Skilled" and "Intelligent" and less "Repetitive".

The superiority of conditions A and D is corroborated by a statistical analysis of variance of the questionnaire answers. Using two-tailed t-tests with Holm correction, these conditions can be compared to condition C on each of the scales under consideration.

Table 5-3 shows the results of this analysis. These shows the significant advantage of the interaction produced by our framework over the control condition both in terms of IMI Interest and Realism. It also reaffirms the finding that "Skilled", "Aggressive" and "Intelligent" are adjectives associated with conditions A and D, whereas C is more typically "Clumsy" and "Repetitive".

In spite of the positive outcome, the limitations of a non-interactive study like this must be acknowledged. Indeed, several participants commented on the difficulty to assess a VR experience through a video recording. While it facilitates recruitment and provides feedback from a more diverse pool of participants, the fact is that a full evaluation of the framework requires personal interaction with the system, which leads us to our second study.

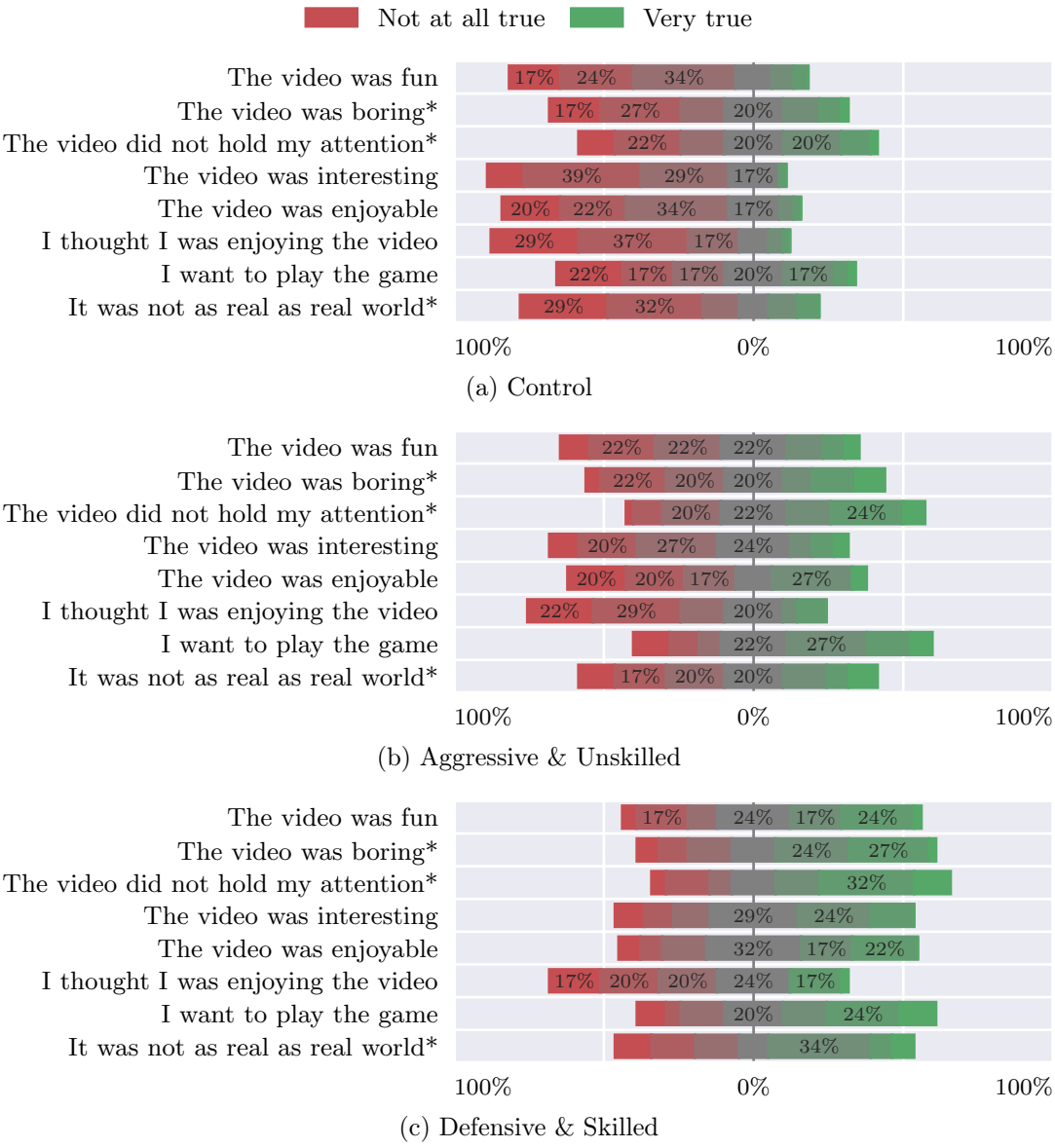


Figure 5-9: Results from the Intrinsic Motivation and Immersion items from each condition in the questionnaire study. Ratings for items marked with * are reversed.

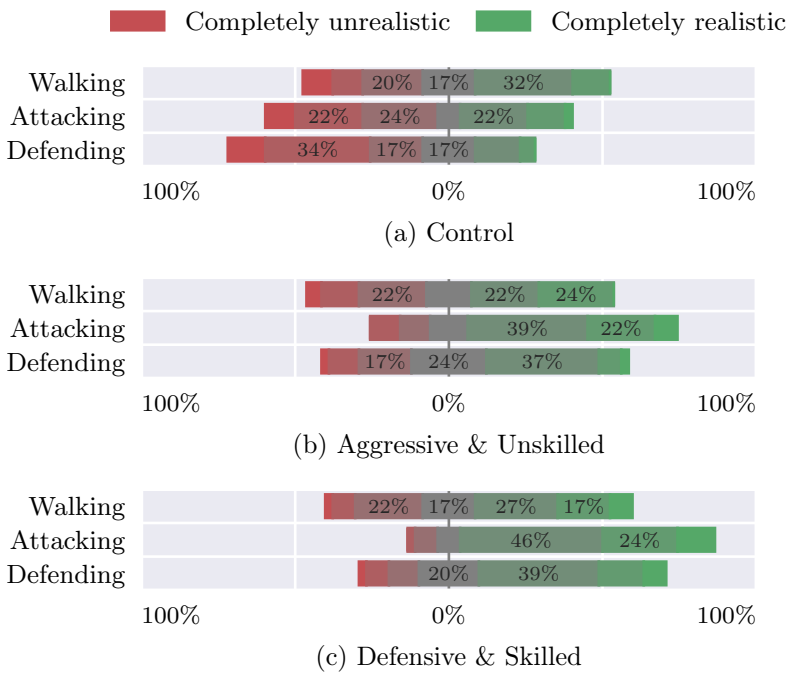


Figure 5-10: Results from the Realism items from each condition in the questionnaire study.

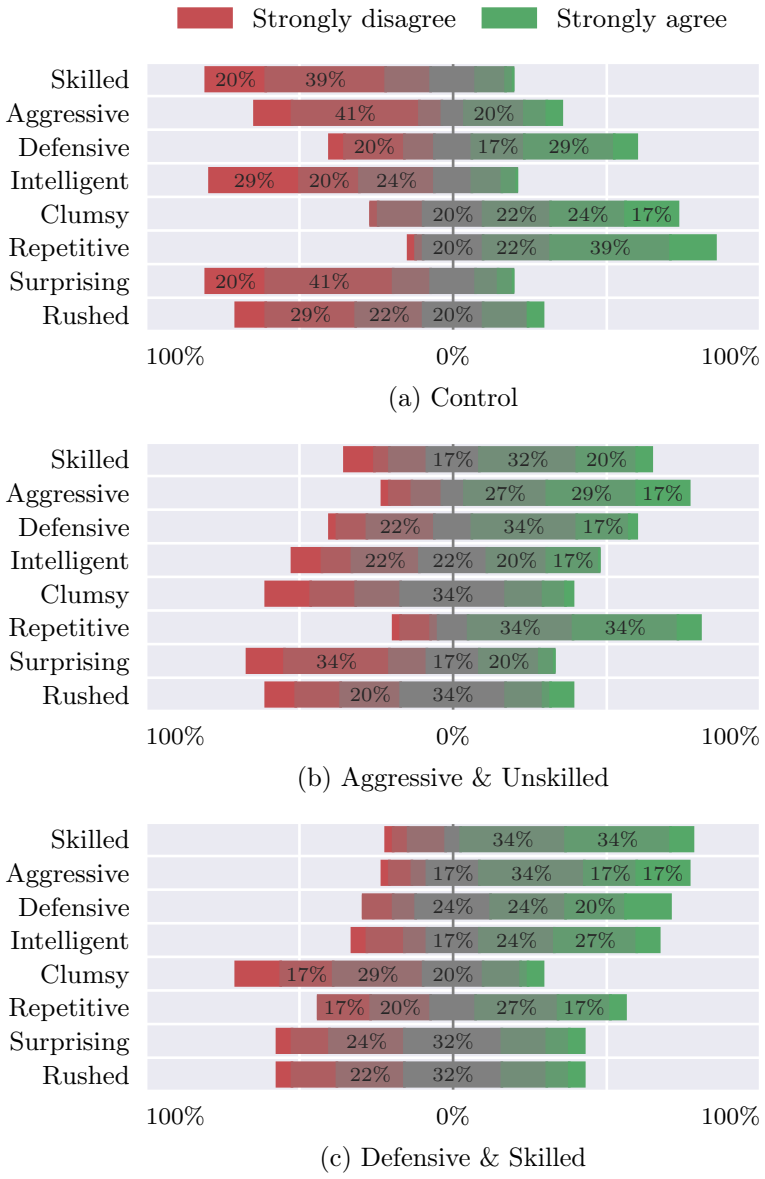


Figure 5-11: Results from the descriptive items from each condition in the questionnaire study.

5.7.2 Interactive Study

Following up from the results of the questionnaire study, the goal of the interactive study is to confirm our findings with users that have personally experienced the interaction with the framework. In this case, participants were asked to interact with each of the proposed conditions in the VR environment. Each interaction was limited to three minutes, as longer sessions of continuous emulated sword fighting were too exhausting for some participants. After each condition, participants completed the same questionnaire as in the previous study, with adapted wording (e.g. "Playing the game was fun" instead of "Watching the video was fun"). They were also asked to complete the Witmer and Singer Presence Questionnaire (1998), a collection of 30 items measuring the sense of "being there" in the environment evaluated in seven-point Likert scales.

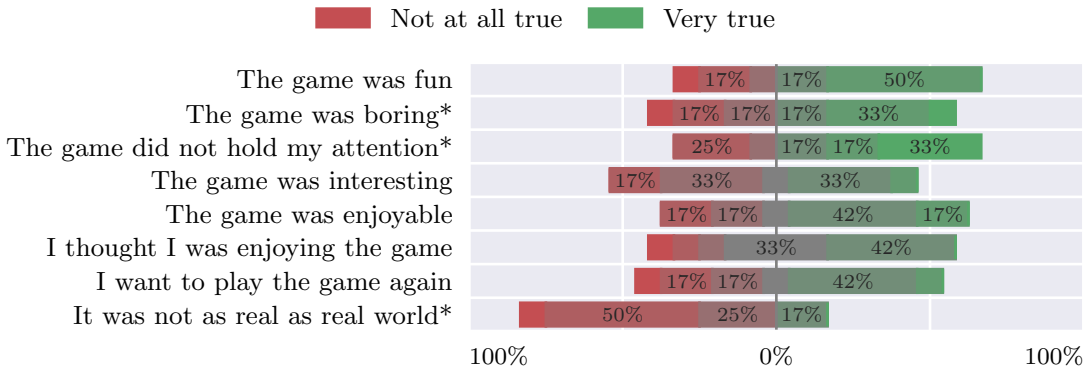
The sample for this study was of twelve participants, aged between 23 and 40 (mean 31.9, s.d. 5.5) and split into four female and eight male participants. Their mean self-assessed experience, in a scale from one to seven, with video games was 4.25 (s.d. 0.75); with virtual reality, 3.33 (s.d. 0.89); with video games development, 3.58 (s.d. 1.00); with 3D animation, 2.83 (s.d. 0.84); and with actual sword fighting, 1.67 (s.d. 0.78). Overall, participants had more experience in all relevant areas in this study.

As before, answers from participants are summarised in figs. 5-12 to 5-14 (pages 113 to 115). Unsurprisingly, participants found the interactive experience far more interesting than the videos. However, the control condition C falls significantly behind A and D in this scale and in Realism. The differences in the results for the descriptive terms become even more pronounced here, with condition C being far less "Skilled" or "Intelligent" than A and D. Participants also noted a significant difference in how "Aggressive" conditions A and D were, showing the impact of their different parameterisations.

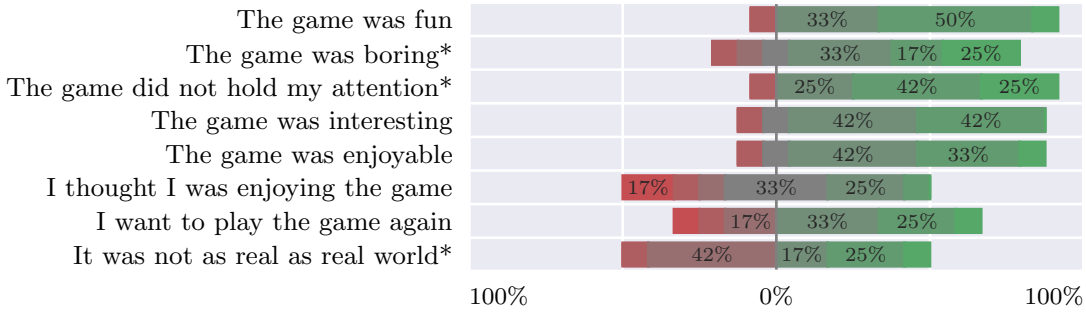
The statistical analysis in table 5-4 backs again the previous discussion. While results are not as strongly significant in this case, which can be attributed to the reduced sample size and the more critical perception of the participants, the same trends found before appear here. It is worth noting that, unlike Interest and Realism, Presence does not seem to be impacted by the condition. This subscale gauges the impression of actually being inside of the virtual environment, and it relates to the quality and consistency of the perceived sensory information. In this case, this appears to be determined primarily by the environment itself, shared across all conditions, rather than the character behaviour. As shown above, the descriptive terms point again to a more positive perception of conditions A and D in terms of intelligence and skill. All these factors suggest a more enjoyable and fulfilling interactive experience for users of our framework.

Scale	Cond. 1	Cond. 2	$t(11)$	p
IMI Interest	A	C	2.353	0.077
	D	C	2.833	0.049*
Realism	A	C	3.352	0.019*
	D	C	3.120	0.019*
Presence	A	C	1.129	0.566
	D	C	2.636	0.069
Skilled	A	C	5.046	0.001*
	D	C	3.339	0.013*
Aggressive	A	C	5.564	< 0.001*
	D	C	2.106	0.059
Defensive	C	A	1.131	0.846
	D	C	0.561	0.846
Intelligent	A	C	3.350	0.019*
	D	C	2.919	0.028*
Clumsy	C	A	2.449	0.097
	C	D	2.184	0.103
Repetitive	C	A	0.804	0.657
	C	D	1.995	0.214
Surprising	A	C	0.586	1.000
	D	C	1.292	0.669
Rushed	A	C	0.714	1.000
	D	C	0.646	1.000

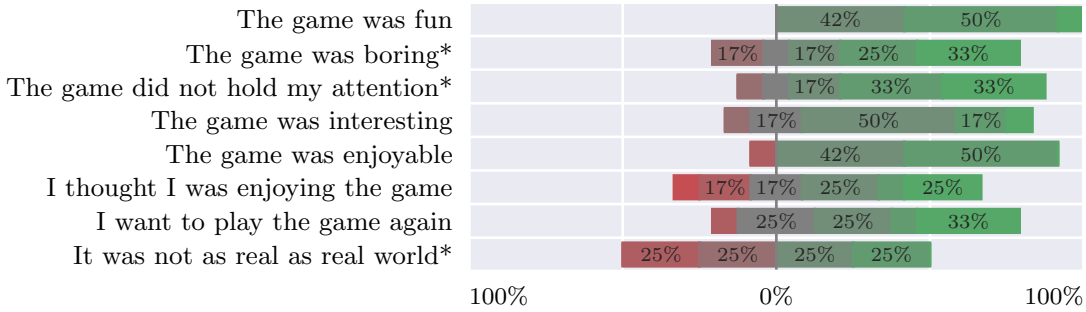
Table 5-4: One-tailed t-tests over the results of the interactive study. The hypothesis under test is that the measure for condition 1 is greater than that for condition 2. Starred rows indicate statistical significance ($p < 0.05$).



(a) Control



(b) Aggressive & Unskilled



(c) Defensive & Skilled

Figure 5-12: Results from the Intrinsic Motivation and Immersion items from each condition in the interactive study. Ratings for items marked with * are reversed.

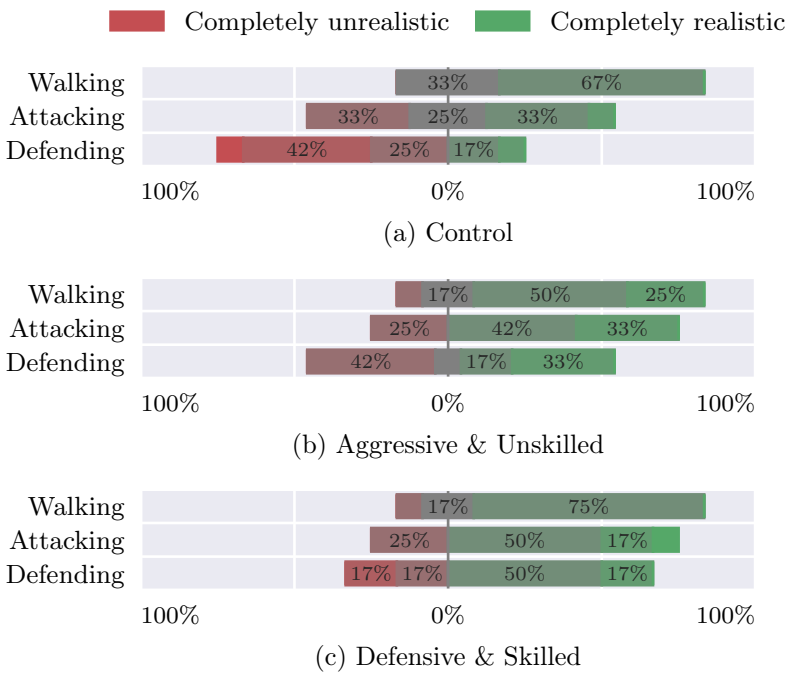


Figure 5-13: Results from the Realism items from each condition in the interactive study.

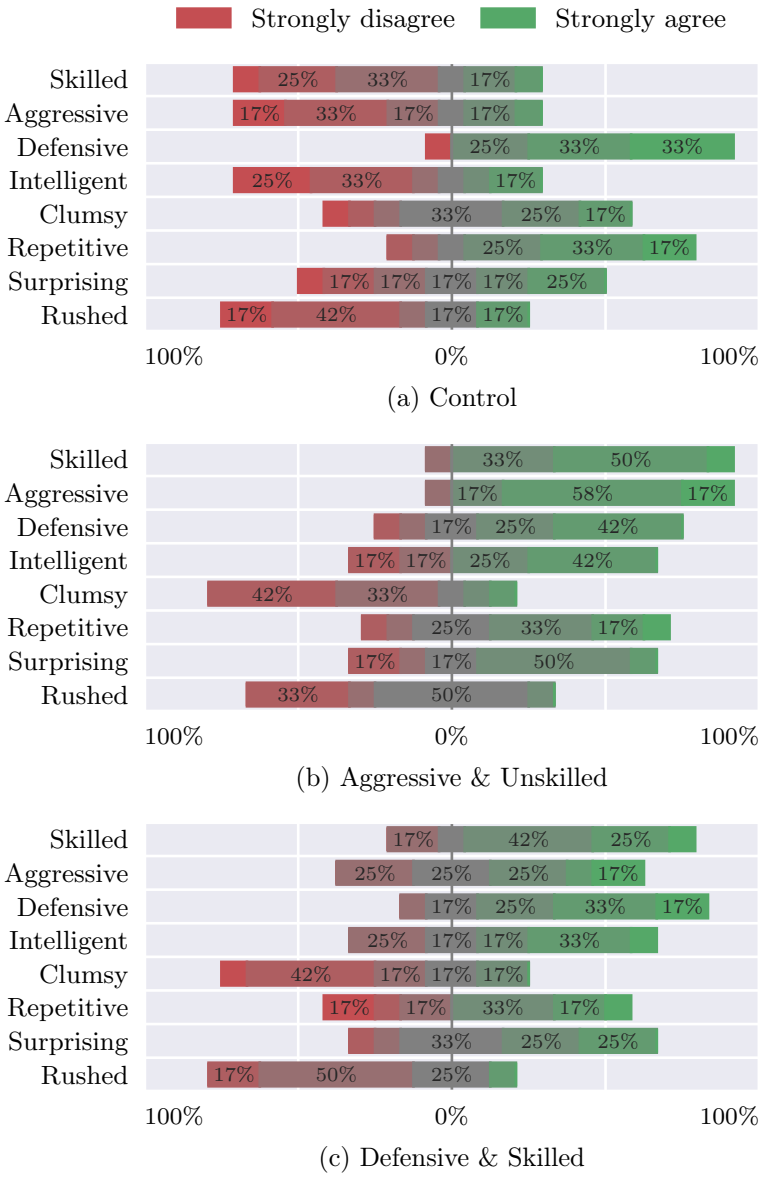


Figure 5-14: Results from the descriptive items from each condition in the interactive study.

5.8 Discussion

This chapter presents a novel data-driven framework for human–computer interaction in VR, a medium where the application of many conventional animation and design techniques becomes difficult or impractical. Modelling the process of interaction as a whole, the strength of the framework lies in the mixture of tools and approaches it brings together. The division of the problem into a physical and a semantic plane clearly defines what parts of the process should or should not be automated, establishing the barrier between data-driven models and handmade logic. This is key to the predictability of the interaction, and puts boundaries to what needs to be understood in every detail, leaving the rest to be enclosed in a black box. Breaking the solution into three components allows us to analyse each different part with relative independence from each other. This makes it possible to easily integrate the animation synthesis techniques presented in chapter 4 along with a gesture recognition model and behavioural logic.

The sword fighting scenario developed throughout our exposition demonstrates the potential of our framework in a realistic use case. The ideas introduced here, though, can be applied to any sort of interaction. Each particular context would of course require an analysis of its specific needs. But, with regard to our methodology, designers would just need to define the vocabulary of gestures of interest, possibly adjust the selection of features in the data-driven models and, most importantly, design the behaviour to implement. These are all differences in the specific data and configuration of the model; the overall scheme of things remains sound and general enough to be widely applicable.

A significant part of this chapter is dedicated to evaluation of one kind or another. Certainly, evaluating such a system is a difficult task for multiple reasons, including the specificity of the problem (for whichever particular scenario one may choose), the multiple elements involved and the subjective quality of the user perception. We found it effective to combine the numerical evaluation of the data-driven models with user studies to complete a comprehensive view of the quality of the system. On the one hand, we can rely on the performance of the models making up the individual parts of the physical level, which establishes confidence on the less transparent aspects of the system. On the other hand, results from our user study tell us that our framework is a strong alternative to traditional interaction design techniques.

As a whole, the benefits of our framework are both a simplified development methodology, requiring less manual animation and input processing work, and a more enjoyable and engaging experience for the end users, who perceive an interaction with the VR environment that is more natural and realistic.

Chapter 6

Conclusion

This thesis introduces new perspectives and insights in the areas of real-time animation and interaction, proposing methods that support a new data-driven approach to the development of VR experiences and animated characters in general. It builds on a series of recent works (Holden, Komura and Saito, 2017; Zhang et al., 2018; Starke, Zhang et al., 2019) that signal the interest of both academia and industry in smart tools that can alleviate the workload and expand the capabilities of animators and designers, through intelligent use of the massive volumes of data and computational power afforded by current technology. Let us recollect the obtained results and situate them in a broader context.

6.1 Research Output

We began this work with two distinct goals in mind. First, the development of a new machine learning model for real-time character animation synthesis that addresses the shortcomings of prior proposals and can be easily adapted to new situations. Second, the definition of a new methodology to develop interactive VR scenarios that leverages data-driven techniques to improve the player experience and reduce the amount of human work. To that end, this thesis introduced the following.

- A new kind of neural network architecture, grid-functioned neural networks (GFNN), that targets some of the specific needs of animation synthesis. In particular, it makes it possible to specialise different parts of a model in different aspects of the animation, overcoming the limitations of general monolithic models. More generally, GFNN subdivides the domain of the problem across an arbitrary number of

dimensions, allowing it to learn local patterns that better capture the variety of highly nonlinear data. The presented evaluation shows the advantages of GFNN over regular neural networks in terms of accuracy, performance and scalability for a set of synthetic problems.

- An application of GFNN to the specific animation problem of quadruped locomotion, including the design of the intrinsic model grid and the necessary preprocessing steps to make the most of the available data. The results yielded by the comparison with another recent proposal, the MANN model (Zhang et al., 2018), highlight the tighter control and better predictability of GFNN, with no impact to the perceived realism of the generated animation.
- A novel data-driven framework for human–character interaction in VR that takes a comprehensive approach to the problem and offers developers a foundation to fabricate new experiences and video games. Abstracting common concerns into simple and expressive concepts, namely gesture detection, behaviour design and animation synthesis, the threefold structure of the framework clearly separates responsibilities and demarcates the boundary between human-designed logic and data-driven processing.
- A practical case study of this framework for a VR sword fighting scenario, demonstrating how each of its elements take concrete form and come together in a complete interactive experience. A GFNN model is used here again to deal with a very different animation problem, showing the flexibility of its design. The extensive evaluation of the results, including performance metrics and user studies, confirm the potential of our proposal and the value of the introduced methods.

These contributions, along with their associated results,[1] are significant advances of the state of the art that open up new avenues of research in the intersection of digital entertainment and machine learning.

6.2 Impact

Our research can be framed within the blooming field of data-driven techniques and applications in computer science, largely supported by the groundbreaking advances in neural networks published during the last decade. The ambition for algorithms and models capable of eliminating the human factor from costly or complicated processes and tasks has permeated all sorts of disciplines, and digital entertainment is no exception. From texture super-resolution to automated asset generation, machine learning

[1] Code and data available at `https://doi.org/10.15125/BATH-00752` (Dehesa, 2020).

has found all sorts of applications just within the domain of video game development.

Character animation is one of the areas that can benefit the most from this trend. As virtual worlds become larger and real-time computer graphics get ever closer to reality, the scale and quality demanded from animation systems are pushing the limits of traditional methods. Eventually, the search for new approaches that offload at least part of the task to an autonomous algorithm becomes less of a speculation and more of a necessity. The case study presented in chapter 5 is exemplary of the simplified view of complex problems that data-driven solutions can support and the measurable benefits of the results they yield.

The industrial environment in which this research was carried out further reinforces the importance and utility of this perspective. Developed in cooperation with leading industrial experts, our contributions are designed to become part of the animation toolkit of high-quality video game productions. In the case of our partner studio Ninja Theory, our results have already impacted their approach to animation in current projects, helping shape a new workflow that integrates data-driven models to support the development of upcoming titles.

6.3 Future Work

The tools and techniques introduced here leave the door open to the research of a number of questions that escape the scope of this work. With respect to the GFNN architecture, in spite of our thorough characterisation, the full extent of its potential is yet to be explored. While we have shown positive results in small synthetic problems and in animation scenarios, it would be interesting to study its application to other domains, and specifically those where neural networks are already popular, like computer vision or language processing. This would highlight the advantages and drawbacks of GFNN when compared to other mainstream architectures, such as convolutional or recurrent networks, and the possible benefits of combining these approaches. Furthermore, another compelling research topic is the design of other expert structures as alternatives to a grid. This could result in more suitable models for unevenly distributed datasets, and even open the possibility of an adaptive kind of model with a dynamic number and structure of experts.

With respect to the VR interaction framework, there are several matters susceptible to further research. While the conceptual design is sound on its own, its evaluation is limited to the presented case study. A second case study would give a clearer picture of the application of the framework and the differences and commonalities that arise

between different cases. Even within the analysed case, an extension of its scope, for example to include a walking opponent, or sword fighting with a shield, would shed light on the necessary changes and adaptations that a developer might be expected to carry out. The choice of a state machine for behaviour planning is, for all its convenience, limited too, and the incorporation of more sophisticated reasoning models would be worthy of investigation. The evaluation of the results is also an open question, since both performance metrics and user studies are not entirely straightforward to implement and interpret. We expect this to be a topic of growing interest as the popularity of data-driven methods in human–computer interaction keeps rising.

6.4 Closing Comments

It is an exciting time for the field of digital entertainment. The confluence of formidable advances in hardware, computer graphics, machine learning and other related disciplines puts researchers and practitioners in a privileged position to build and experiment with a plethora of ideas that were, until not too long ago, simply infeasible. The body of research that is being produced today will be the basis of a new generation of not just technologies, but entire ways of working and producing content that put data at their centre.

Our research adds a new stepping stone along this path. It makes concrete advances, specifically in the areas of real-time animation and interaction, that bring us closer to that future. The results provided by these new methods anticipate the potential that data-driven techniques hold for this field, and the new paradigms that will be shaped by them.

Appendix A

Neural Networks Background

This appendix collects common results and formulas that are used in the design and training of neural networks. The ideas reviewed here are applied in chapters 3 to 5 in the different experiments carried out as part of our work. Relevant concepts for each experiment are indicated within the corresponding discussion in each case.

A.1 Optimisation

The process of training a training a neural network is, at its core, no more than the optimisation of a non-linear differentiable function. Let us consider the problem of regressing an unknown function $F\colon \mathbb{R}^N \to \mathbb{R}^M$. A neural network can then be simply represented as a function $G\colon \mathbb{R}^N \times \mathbb{R}^T \to \mathbb{R}^M$ takes a given input vector $\boldsymbol{x} \in \mathbb{R}^N$ and a vector of trainable parameters $\boldsymbol{\theta} \in \mathbb{R}^T$ (these can be thought of as a concatenation of all the randomly initialised weights and biases of the neural network) to produce an output $\widehat{\boldsymbol{y}} = G(\boldsymbol{x}, \boldsymbol{\theta})$ that approximates $\boldsymbol{y} = F(\boldsymbol{x})$. The goal of the training is then to minimise an objective or "loss" function $L\colon \mathbb{R}^M \times \mathbb{R}^M \to \mathbb{R}$ that represents, in an abstract sense, the difference between the actual function value $\boldsymbol{y}$ and the model prediction $\widehat{\boldsymbol{y}}$ (more generally, a differentiable function that decreases as $\widehat{\boldsymbol{y}}$ approaches $\boldsymbol{y}$). In a simple case, this can just be the squared norm $\|\boldsymbol{y} - \widehat{\boldsymbol{y}}\|^2$, though different problems may benefit from different choices. The key insight here is that it is possible to compute the gradient of the loss value with respect to the parameters vector $\boldsymbol{\theta}$:

$$\nabla_{\boldsymbol{\theta}} L(\boldsymbol{y}, \widehat{\boldsymbol{y}}) = \{\partial_{\theta_1} L(\boldsymbol{y}, \widehat{\boldsymbol{y}}), \ldots, \partial_{\theta_T} L(\boldsymbol{y}, \widehat{\boldsymbol{y}})\} \tag{A.1}$$

And, by definition, the value of $L(\boldsymbol{y}, \widehat{\boldsymbol{y}})$ can be reduced by shifting $\boldsymbol{\theta}$ in opposite direction to $\nabla_{\boldsymbol{\theta}} L(\boldsymbol{y}, \widehat{\boldsymbol{y}})$. This is precisely the principle behind the gradient descent algorithm (Cauchy, 1847; Bottou, 1998), which simply applies the following formula iteratively:

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \mu \nabla_{\boldsymbol{\theta}} L(\boldsymbol{y}, \widehat{\boldsymbol{y}}) \tag{A.2}$$

Where $\mu \in \mathbb{R}^+$ is a "learning rate" that must be manually tuned so as to advance the training effectively without taking excessively large steps on each iteration that take the model further away from the minimal configuration. This is repeated until some stopping condition is met (in our experiments, after a particular number of iterations is completed).

Gradient descent, while simple, suffers from a number of issues. In the first place, the learning rate needs to be adjusted per case, largely by trial and error. Moreover, it is frequently the case that, in the first stages of the training, a larger learning rate facilitates the exploration of the search space, while latter steps benefit from a small rate that precisely takes the model parameters to their optimal value. In addition to that, using only information from the local gradient is limiting as well, as it can sometimes momentarily take the model in the wrong direction (similar to water running down an irregular slope).

Several improvements on this scheme have been proposed to address its shortcomings. One of the most popular developments is the Adam optimisation algorithm (Kingma and Ba, 2015). It attempts to model the "inertia" of the optimisation process by maintaining a running average of the gradient, $\boldsymbol{m} \in \mathbb{R}^T$, and of its second moment, $\boldsymbol{v} \in \mathbb{R}^T$ (both initialised to a null vector), as well as a count of the number of optimisation steps $t \in \mathbb{N}$. Each iteration runs then as follows (operations on vectors are component-wise):

$$\begin{aligned}
t &\leftarrow t + 1 \\
\boldsymbol{m} &\leftarrow \beta_1 \boldsymbol{m} + (1 - \beta_1) \nabla_{\boldsymbol{\theta}} L(\boldsymbol{y}, \widehat{\boldsymbol{y}}) \\
\boldsymbol{v} &\leftarrow \beta_2 \boldsymbol{v} + (1 - \beta_2) \left(\nabla_{\boldsymbol{\theta}} L(\boldsymbol{y}, \widehat{\boldsymbol{y}})\right)^2 \\
\widehat{\boldsymbol{m}} &= \frac{\boldsymbol{m}}{1 - \beta_1^t} \\
\widehat{\boldsymbol{v}} &= \frac{\boldsymbol{v}}{1 - \beta_2^t} \\
\boldsymbol{\theta} &\leftarrow \boldsymbol{\theta} - \mu \frac{\widehat{\boldsymbol{m}}}{\sqrt{\widehat{\boldsymbol{v}}} - \epsilon}
\end{aligned} \tag{A.3}$$

Where $\beta_1 \in \mathbb{R}$, $\beta_2 \in \mathbb{R}$ and $\epsilon \in \mathbb{R}$ are parameters of the algorithm (in addition to the previously existing learning rate μ). It may seem that this algorithm is even less usable due to the additional parameters, but the authors propose some recommended values, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$, that rarely need adjusting. The learning rate can also be typically set to $\mu = 0.001$, and may require only minor tuning. Adam is currently one of the most widespread optimisation algorithms used in neural networks research and development due to its simplicity of use and reliability, and is the one used in all of our experiments.

A.2 Activation and Loss Functions

A key element of neural networks is the non-linear activation function applied to the output of each layer in the model. Every layer output vector $\boldsymbol{x} \in \mathbb{R}^N$ is passed through such a function to produce a new vector $\boldsymbol{x'} \in \mathbb{R}^N$ that is received by the next layer. This is an essential feature, since otherwise the neural network would be equivalent to a simple linear mapping (that is, any neural network would be reducible to a single layer). The only exception is the output layer of the network, which can sometimes have no activation, also called "linear" activation, $\boldsymbol{x'} = \boldsymbol{x}$, in order to model an unbounded value (in which case data normalisation is advisable). In those cases, the most frequent choice for the loss function with respect to the expected value $\boldsymbol{y} \in \mathbb{R}^N$ is simply the sum of squared differences between the two vectors, $\|\boldsymbol{x'} - \boldsymbol{y}\|^2$.

One common option for activation is the sigmoid function, defined as (operations are component-wise):

$$\text{Sigmoid}(\boldsymbol{x}) = \frac{e^{\boldsymbol{x}}}{e^{\boldsymbol{x}} + 1} \tag{A.4}$$

The sigmoid function effectively "squashes" its input to the range $(0, 1)$, which makes it suitable to model binary problems, like binary classification. When used in this way, it is common to use the cross-entropy of $\boldsymbol{x'}$ and $\boldsymbol{y}$ as training loss:

$$\text{Cross-entropy}(\boldsymbol{x'}, \boldsymbol{y}) = -\sum_{i=1}^{N} y_i \log(x'_i) + (1 - y_i) \log(1 - x'_i) \tag{A.5}$$

In the case of multi-class classification problems, though, it may be preferred to apply a softmax activation function to the output layer of the model, defined as:

$$\text{Softmax}(\boldsymbol{x})_i = \frac{e^{x_i}}{\sum_{j=1}^{N} e^{x_j}} \tag{A.6}$$

While the softmax function also has the property of squashing its inputs to $(0, 1)$, it has the additional quality that the sum of its outputs is one, which is analogous to a categorical probability distribution. A softmax activation at the output of a neural network is best paired with a softmax cross-entropy loss function, which is simply:

$$\text{Softmax cross-entropy}(\boldsymbol{x}', \boldsymbol{y}) = -\sum_{i=1}^{N} y_i \log(x'_i) \tag{A.7}$$

In a sense, the softmax and softmax cross-entropy functions generalise the probabilistic interpretation of the sigmoid function, which assigns its value to the probability of one class and the implicit complementary value (subtracted from one) to another class, to an arbitrary number of classes.

Many other functions have been proposed as activation functions. Another popular option is the hyperbolic tangent, which just extends the range of the sigmoid function to $(-1, 1)$, and hence has similar properties to it:

$$\tanh(\boldsymbol{x}) = 2\,\text{Sigmoid}(\boldsymbol{x}) - 1 \tag{A.8}$$

However, other activation functions have been designed to address specific issues in the design of neural networks. Specifically, the sigmoid function (and related) are prone to what is known as the "vanishing gradient" problem (Hochreiter, 1998). This happens when the backpropagation of a gradient through a long chain of operations (which is common in very deep models or recurrent networks) becomes nearly zero due to the consecutive multiplication of values smaller than one. While the activation function is not the only cause for this (and, in fact, the opposite "exploding gradient" problem may occur in different circumstances), some choices can mitigate it. Also, the sigmoid function is relatively expensive in computational terms. With this in mind, and considering that, according to the theory, any non-linear function can work as an activation function, Nair and Hinton, (2010) proposed the rectified linear unit activation, or ReLU:

$$\text{ReLU}(\boldsymbol{x})_i = \max(x_i, 0) \tag{A.9}$$

This is an extremely simple function, not too different from a linear activation, that nonetheless has been proved to be an effective activation function for hidden layers in many different problems and model architectures.

There exists a wide variety of activation functions and loss functions in the literature. The ones reviewed here cover the design of the models and experiments in our work, but the interested reader can consult the surveys on this topic by Nwankpa et al., (2018) and Janocha and Czarnecki, (2017).

A.3 Regularisation and Weight Decay

As a general concept of machine learning (or, more broadly, function fitting), regularisation refers to measures or techniques that are used in the construction of a supervised model to prevent it from overfitting to the available data. Put simply, overfitting is the phenomenon that occurs when a model yields very good predictions for data points in the training dataset but very bad ones for those out of it. It usually affects models with large absolute values in their parameters, since those tend to result in a less smooth or stable behaviour (that is, two similar inputs produce very dissimilar outputs), and so regularisation attempts to constraint the scale of these parameters.

Neural networks are prone to overfitting, and can usually improve their performance with regularisation. The most common method to do this is what is known as "weight decay", which simply alters the optimisation objective by adding a factor that depends on the aggregated absolute parameter values. In the case of ℓ^2 regularisation, the squared norm of the parameters vector is used. Given a neural network with parameters $\boldsymbol{\theta} \in \mathbb{R}^T$, a function value $\boldsymbol{y} \in \mathbb{R}^M$, a prediction $\widehat{\boldsymbol{y}} \in \mathbb{R}^M$ and a loss value $L(\boldsymbol{y}, \widehat{\boldsymbol{y}}) \in \mathbb{R}$, a new objective function $L' \colon \mathbb{R}^M \times \mathbb{R}^M \to \mathbb{R}$ is defined as:

$$L'(\boldsymbol{y}, \widehat{\boldsymbol{y}}) = L(\boldsymbol{y}, \widehat{\boldsymbol{y}}) + \lambda \|\boldsymbol{\theta}\|^2 \tag{A.10}$$

Where $\lambda \in \mathbb{R}^+$ is a regularisation parameter that must be adjusted depending on the model complexity and the problem.

Weight decay is very commonly used for all types of neural networks, usually with significant benefits. However, it can be difficult to balance its importance in the training, as excessive regularisation will prevent the model from learning properly. Also, neural networks with a very large number of parameters can be difficult to regularise in this manner. For the purposes of our work, though, weight decay is an effective way to

regularise our models.

A.4 Dropout

Dropout (Srivastava et al., 2014) is another category of regularisation strategy for neural networks. It is based on the idea (biologically inspired, to some extent) that a neural network will be more resilient if it learns redundant patterns. What this would mean is that multiple units in the network would be activated under similar circumstances, and so if one particular unit fails to be activated as it should under some input, then another one would supply that fault so the model still produces the correct output. The goal of dropout is to encourage this characteristic by randomly suppressing connections in the neural network during training (that is, temporarily setting the weight and bias corresponding to a connection to zero). Given a neural network layer with activation values $\boldsymbol{x'} \in \mathbb{R}^N$, the dropout vector received by the i-th unit in the next layer $\widetilde{\boldsymbol{x}}^i \in \mathbb{R}^L$ is defined as:

$$\widetilde{x}^i_j = \begin{cases} \frac{x'_j}{1-\rho} & \text{if } r^i_j \geq \rho \\ 0 & \text{otherwise} \end{cases} \tag{A.11}$$

Where $\rho \in [0, 1)$ is the dropout rate and each $r^i_j \sim \mathcal{U}(0, 1)$ is a random value that determines whether the connection is dropped or not. A dropout rate of zero would cancel the effect of dropout, while a rate approaching one would drop most connections in the network. The remaining connections are scaled up by a factor of $1/(1-\rho)$, so the scale of the aggregated activation values received by the next layer remains unaffected.

Dropout has a strong regularisation effect, it impacts the training process of a neural network significantly increasing the required number of optimisation steps. However, it can be an effective method for models with many layers where weight decay does not produce good results.

Appendix B

Angles and Rotations

This appendix presents some geometrical concepts and results that are used at different points in our research. We assume basic understanding of 3D geometry and affine transformations.

B.1 3D Rotation Representations

Tridimensional rotations can be expressed through a variety of mathematical representations with different properties, advantages and disadvantages. 3D rotations have three degrees of freedom, which can be though of as the angle of rotation around each axis, since any arbitrary rotation is always equivalent to three consecutive rotations around orthogonal axes; however, representations in higher-dimensional spaces are frequently found to be more useful. When operating with 3D data, such as elements of $\mathbb{R}^3$, a rotation can be most conveniently expressed as an orthogonal 3×3 matrix (specifically, proper rotations correspond to those with a determinant of 1). This allows to compute the rotated position of a point through a simple vector–matrix product. Nonetheless, rotation matrices are not an economical representation of rotations, containing nine different values. The columns of the rotation matrix indicate the directions of the three rotated axes, but other geometrical aspects, like the axis or angle of rotation, are not immediately apparent. Also, while two rotation matrices can be composed via matrix product, they cannot be easily interpolated in a meaningful sense.

As suggested above, a rotation can be intuitively expressed as three angles of rotation around the axes, usually called Euler angles. There are several conventions about how these are interpreted, depending on the axes layout and the order in which they are

applied, but all of them are equivalent. Assuming the x axis is the forward direction, y is the right direction and z is the up direction, a common nomenclature for these three angles is: yaw (ψ), for the rotation around z; pitch (θ), for the rotation around y; and roll (ϕ), for the rotation around x.

These can be readily converted into a rotation matrix computed as:

$$\begin{pmatrix} \cos\phi & -\sin\phi & 0 \\ \sin\phi & \cos\phi & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\psi & -\sin\psi \\ 0 & \sin\psi & \cos\psi \end{pmatrix} \tag{B.1}$$

Unfortunately, some properties of Euler angles render them unsuitable for many uses. On the one hand, even restricting the angle values to $-\pi$ to π radians, some 3D rotations can be represented by more than one triple of angles. On the other hand, they suffer from the "gimbal lock" problem, which is, in simple terms, the loss of one degree of freedom in the rotation under particular circumstances, namely when one axis is rotated onto one of the subsequent rotation axes. They are also not directly useful for most geometrical operations, requiring conversion into some other form.

Rotations can also be expressed as a unit quaternion. A quaternion $\boldsymbol{q}$ is a number composed of a real part and an imaginary part on the basis of three imaginary units but, for the purposes of this exposition, it can be described as a tuple of four numbers $(q_x, q_y, q_z, q_w) \in \mathbb{R}^4$. Its norm is defined analogously to the Euclidean norm:

$$\|\boldsymbol{q}\| = \sqrt{q_x^2 + q_y^2 + q_z^2 + q_w^2} \tag{B.2}$$

And thus a unit quaternion is one such that $\|\boldsymbol{q}\| = 1$, and it can be used as a representation of the 3D rotation given by the matrix:

$$\begin{pmatrix} 1 - 2(q_y^2 + q_z^2) & 2(q_x q_y - q_z q_w) & 2(q_x q_z + q_y q_w) \\ 2(q_x q_y + q_z q_w) & 1 - 2(q_x^2 + q_z^2) & 2(q_y q_z - q_x q_w) \\ 2(q_x q_z - q_y q_w) & 2(q_y q_z + q_x q_w) & 1 - 2(q_x^2 + q_y^2) \end{pmatrix} \tag{B.3}$$

The quaternion corresponding to a rotation with angle $\alpha \in [-\pi, \pi]$ around an axis in direction of the unit vector $\boldsymbol{v} \in \mathbb{R}^3$ can as well be computed as:

$$
\begin{aligned}
q_x &= v_x \sin\frac{\alpha}{2} \\
q_y &= v_y \sin\frac{\alpha}{2} \\
q_z &= v_z \sin\frac{\alpha}{2} \\
q_w &= \cos\frac{\alpha}{2}
\end{aligned}
\tag{B.4}
$$

In addition to being smaller than full rotation matrices, quaternions have several geometrical uses. They can be easily composed and inverted, and it is possible to directly and quickly compute the rotated coordinates of a point without constructing the rotation matrix. They can also be used to blend between two given rotations using spherical linear interpolation (Shoemake, 1985), which enforces constant angular speed across the shortest rotational path. However, there is also not a unique correspondence between 3D rotations and quaternions, as the rotation represented by a quaternion $\boldsymbol{q}$ and its negation $-\boldsymbol{q}$ is actually the same. In addition to that, the continuity of quaternions and rotations do not match either; in some cases, two quaternions with similar values correspond to very different rotations, and vice versa. These two characteristics can make it difficult to numerically estimate rotations expressed as quaternions, especially for values close to those discontinuities.

This takes us to the last representation of rotations discussed here, which is a pair of rotated axis vectors. This are the result of applying the rotation to two unit vectors along a pair of orthogonal axes; for example, given a rotation matrix $M \in \mathbb{R}^{3\times3}$, we can take the vectors $\boldsymbol{x} = [1, 0, 0]$ and $\boldsymbol{y} = [0, 1, 0]$ to obtain the pair of rotated axis vectors $\boldsymbol{x}' = M\boldsymbol{x}$ and $\boldsymbol{y}' = M\boldsymbol{y}$.

Note the third rotated axis vector can always be computed as the cross product $\boldsymbol{z}' = \boldsymbol{x}' \times \boldsymbol{y}'$. The corresponding rotation matrix is then simply expressed as:

$$
\begin{pmatrix}
x'_x & y'_x & z'_x \\
x'_y & y'_y & z'_y \\
x'_z & y'_z & z'_z
\end{pmatrix}
\tag{B.5}
$$

Comprising a total of six values, this is not as small as other representations. However, each pair of vectors univocally corresponds to a rotation, and it does not exhibit any sort of discontinuity. It has similar drawbacks to the full rotation matrix, but it is well suited to numerical approximation.

B.2 Spherical Coordinates

Spherical coordinates are an alternative to Cartesian coordinates to encode 3D positions in space. They are derived from the angles in a sphere centred at the origin and containing the position. Given a point $\boldsymbol{p} \in \mathbb{R}^3$, the corresponding spherical coordinates are defined as:

$$\begin{aligned} \rho &= \|\boldsymbol{p}\| \\ \theta &= \arccos \frac{p_z}{\|\boldsymbol{p}\|} \\ \phi &= \arctan \frac{p_y}{p_x} \end{aligned} \tag{B.6}$$

Where $\rho \in \mathbb{R}^+$ is called the radius, $\theta \in [-\pi, \pi)$ is the inclination, which is the angle between the up direction and $\boldsymbol{p}$, and $\phi \in [-\pi, \pi)$ is the azimuth, which is the angle between the forward direction and the projection of $\boldsymbol{p}$ over the horizontal plane.

Spherical coordinates offer a different perspective of the geometry of a scene, and in particular of the relation between different positions, and so they can be a useful input to a geometrical processing function or algorithm.

Bibliography

Ackley, D.H., 1987. *Stochastic Iterated Genetic Hillclimbing.* Pittsburgh, PA, USA: Carnegie Mellon University.

Antoniades, T., Libreri, K., Caulkin, S. and Mastilovic, V., 2016. Creating a Live Real-time Performance-captured Digital Human. *ACM SIGGRAPH 2016 Real-Time Live! (SIGGRAPH '16)*, 24–28 July 2016, Anaheim, CA, USA. ACM, 36:21–36:21.

Asadi-Aghbolaghi, M., Clapés, A., Bellantonio, M., Escalante, H.J., Ponce-López, V., Baró, X., Guyon, I., Kasaei, S. and Escalera, S., 2017. Deep Learning for Action and Gesture Recognition in Image Sequences: A Survey. *Gesture Recognition*, The Springer Series on Challenges in Machine Learning. Springer, pp.539–578.

Bailenson, J.N., Blascovich, J., Beall, A.C. and Loomis, J.M., 2003. Interpersonal Distance in Immersive Virtual Environments. *Personality and social psychology bulletin*, 29(7), pp.819–833.

Bartneck, C., Kulić, D., Croft, E. and Zoghbi, S., 2009. Measurement Instruments for the Anthropomorphism, Animacy, Likeability, Perceived Intelligence, and Perceived Safety of Robots. *International journal of social robotics*, 1(1), pp.71–81.

Batras, D., Guez, J., Jégo, J.-F. and Tramus, M.-H., 2016. A Virtual Reality Agent-based Platform for Improvisation Between Real and Virtual Actors Using Gestures. *Proceedings of the 2016 Virtual Reality International Conference (VRIC '16)*, 23–25 March 2016, Laval, France. ACM, 34:1–34:4.

Bengio, Y., 2020. *Time to rethink the publication process in machine learning* [Online]. Yoshua Bengio. Available from: `https://yoshuabengio.org/2020/02/26/time-to-rethink-the-publication-process-in-machine-learning/` [Accessed 12 March 2020].

Bergamin, K., Clavet, S., Holden, D. and Forbes, J.R., 2019. DReCon: Data-driven Responsive Control of Physics-based Characters. *Acm transactions on graphics*, 38(6), 206:1–206:11.

Bottou, L., 1998. Online Algorithms and Stochastic Approximations. *Online Learning and Neural Networks*. Cambridge, UK: Cambridge University Press.

Bratman, M., 1987. *Intention, plans, and practical reason*. Cambridge, Mass: Harvard University Press. 200pp.

Broersen, J., Dastani, M., Hulstijn, J., Huang, Z. and van der Torre, L., 2001. The BOID Architecture: Conflicts Between Beliefs, Obligations, Intentions and Desires. *Proceedings of the 4th International Conference on Autonomous Agents (AGENTS '01)*, 28 May–1 June 2001, Montréal, Canada. ACM, pp.9–16.

Bruderlin, A. and Williams, L., 1995. Motion Signal Processing. *Proceedings of the 22nd Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '95)*, 6–11 August 1995, Los Angeles, CA, USA. ACM, pp.97–104.

Büttner, M., 2013. Reinforcement Learning Based Character Locomotion in Hitman: Absolution. Game Developers Conference 2013 (GDC 2013), 25–29 March 2013, San Francisco, CA, USA.

Catmull, E. and Rom, R., 1974. A Class of Local Interpolating Splines. *Computer Aided Geometric Design*. Academic Press, pp.317–326.

Cauchy, A., 1847. Méthode générale pour la résolution des systemes d'équations simultanées. *Comptes rendus de l'académie des sciences*, 25(1847), pp.536–538.

Cheng, H., Yang, L. and Liu, Z., 2016. Survey on 3D Hand Gesture Recognition. *Ieee transactions on circuits and systems for video technology*, 26(9), pp.1659–1673.

Clavet, S., 2016. Motion Matching and The Road to Next-Gen Animation. Game Developers Conference 2016 (GDC 2016), 14–18 March 2016, San Francisco, CA, USA.

Cohen, J., 1977. The t Test for Means. *Statistical Power Analysis for the Behavioral Sciences*. Elsevier, pp.19–74.

Coros, S., Karpathy, A., Jones, B., Reveret, L. and van de Panne, M., 2011. Locomotion skills for simulated quadrupeds. *Acm transactions on graphics*, 30(4), 59:1–59:12.

Crone, S.F., Lessmann, S. and Stahlbock, R., 2006. The impact of preprocessing on data mining: An evaluation of classifier sensitivity in direct marketing. *European journal of operational research*, 173(3), pp.781–800.

De Mulder, W., Bethard, S. and Moens, M.-F., 2015. A survey on the application of recurrent neural networks to statistical language modeling. *Computer speech & language*, 30(1), pp.61–98.

Dehesa, J., 2020. Dataset for "A Novel Neural Network Architecture with Applications to 3D Animation and Interaction in Virtual Reality" [Online]. Bath, UK: University of Bath Research Data Archive. Available from: `https://doi.org/10.15125/BATH-00752`.

Dehesa, J., Vidler, A., Lutteroth, C. and Padget, J., 2019. Towards Data-Driven Sword Fighting Experiences in VR. *Extended Abstracts of the 2019 CHI Conference on Human Factors in Computing Systems (CHI EA '19)*, 4–9 May 2019, Glasgow, UK. Glasgow, UK: Association for Computing Machinery, LBW2117:1–LBW2117:6.

Dehesa, J., Vidler, A., Lutteroth, C. and Padget, J., 2020. Touché: Data-Driven Interactive Sword Fighting in Virtual Reality. *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems (CHI '20)*, 25–30 April 2020, Honolulu, HI, USA, CHI '20. Honolulu, HI, USA: Association for Computing Machinery, pp.1–14.

DeVault, D., Artstein, R., Benn, G., Dey, T., Fast, E., Gainer, A., Georgila, K., Gratch, J., Hartholt, A., Lhommet, M., Lucas, G., Marsella, S., Morbini, F., Nazarian, A., Scherer, S., Stratou, G., Suri, A., Traum, D., Wood, R., Xu, Y., Rizzo, A. and Morency, L.-P., 2014. SimSensei Kiosk: A Virtual Human Interviewer for Healthcare Decision Support. *Proceedings of the 2014 International Conference on Autonomous Agents and Multi-agent Systems (AAMAS '14)*, 5–9 May 2014, Paris, France. International Foundation for Autonomous Agents and Multiagent Systems, pp.1061–1068.

Dixon, P.M., Saint-Maurice, P.F., Kim, Y., Hibbing, P., Bai, Y. and Welk, G.J., 2018. A Primer on the Use of Equivalence Testing for Evaluating Measurement Agreement. *Medicine & science in sports & exercise*, 50(4), pp.837–845.

Elmezain, M., Al-Hamadi, A., Appenrodt, J. and Michaelis, B., 2008. A Hidden Markov Model-based continuous gesture recognition system for hand motion trajectory. *Proceedings of the 19th International Conference on Pattern Recognition (ICPR 2008)*, 8–11 December 2008, Tampa, FL, USA. IEEE, pp.1–4.

Erol, K., Hendler, J.A. and Nau, D.S., 1994. UMCP: A Sound and Complete Procedure for Hierarchical Task-network Planning. In: K.J. Hammond, ed. *Proceedings of the 2nd International Conference on Artificial Intelligence Planning Systems (AIPS '14)*, 13–15 June 1994, Chicago, IL, USA. AAAI, pp.249–254.

Escalera, S., Baró, X., Gonzàlez, J., Bautista, M.A., Madadi, M., Reyes, M., Ponce-López, V., Escalante, H.J., Shotton, J. and Guyon, I., 2014. ChaLearn Looking at People Challenge 2014: Dataset and Results. *Workshops at the 13th European Conference on Computer Vision (ECCV 2014)*, 6–7 September 2014, Zurich, Switzerland, Lecture Notes in Computer Science. Springer, pp.459–473.

Fanello, S.R., Gori, I., Metta, G. and Odone, F., 2017. Keep It Simple and Sparse: Real-Time Action Recognition. *Gesture Recognition*, The Springer Series on Challenges in Machine Learning. Springer, pp.303–328.

Fernández-Redondo, M. and Hernández-Espinosa, C., 2000. A comparison among weight initialization methods for multilayer feedforward networks. *Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks. IJCNN 2000. Neural Computing: New Challenges and Perspectives for the New Millennium.* Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks. IJCNN 2000. Neural Computing: New Challenges and Perspectives for the New Millennium, 27–27 July 2000, Como, Italy. IEEE, 543–548 vol.4.

Fikes, R.E. and Nilsson, N.J., 1971. Strips: A new approach to the application of theorem proving to problem solving. *Artificial intelligence*, 2(3), pp.189–208.

Fitton, I., Finnegan, D.J. and Proulx, M.J., 2020. Immersive virtual environments and embodied agents for e-learning applications. *Peerj computer science*, 6, e315.

Fragkiadaki, K., Levine, S., Felsen, P. and Malik, J., 2015. Recurrent Network Models for Human Dynamics. *Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV '15)*, 7–13 December 2015, Santiago, Chile, pp.4346–4354.

Georgeff, M.P. and Ingrand, F.F., 1989. Decision-Making in an Embedded Reasoning System. *Proceedings of the 11th International Joint Conference on Artificial Intelligence*, 20–25 August 1989, Detroit, MI, USA. Morgan Kaufmann, pp.972–978.

Gleicher, M., 2001. Motion Path Editing. *Proceedings of the 2001 Symposium on Interactive 3D Graphics (I3D '01)*, 19–21 March 2001, Research Triangle Park,NC, USA. ACM, pp.195–202.

Goodfellow, I.J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A. and Bengio, Y., 2014. Generative adversarial nets. *Advances in Neural Information Processing Systems 27*. Curran Associates, Inc., pp.2672–2680.

Heck, R. and Gleicher, M., 2007. Parametric Motion Graphs. *Proceedings of the 2007 Symposium on Interactive 3D Graphics and Games (I3D '07)*, 30 April–2 May 2007, Seattle, WA, USA. ACM, pp.129–136.

Heess, N., TB, D., Sriram, S., Lemmon, J., Merel, J., Wayne, G., Tassa, Y., Erez, T., Wang, Z., Eslami, S.M.A., Riedmiller, M. and Silver, D., 2017. *Emergence of Locomotion Behaviours in Rich Environments* [Online]. arXiv: `1707.02286 [cs]`.

Ho, E.S.L., Chan, J.C.P., Komura, T. and Leung, H., 2013. Interactive Partner Control in Close Interactions for Real-time Applications. *Acm trans. multimedia comput. commun. appl.*, 9(3) (), July, 21:1–21:19.

Ho, E.S.L. and Komura, T., 2011. A finite state machine based on topology coordinates for wrestling games. *Computer animation and virtual worlds*, 22(5) (), 1 September, pp.435–443.

Hochreiter, S., 1998. The Vanishing Gradient Problem During Learning Recurrent Neural Nets and Problem Solutions. *International journal of uncertainty, fuzziness and knowledge-based systems*, 06(02), pp.107–116.

Hochreiter, S. and Schmidhuber, J., 1997. Long Short-Term Memory. *Neural computation*, 9(8), pp.1735–1780.

Holden, D., Kanoun, O., Perepichka, M. and Popa, T., 2020. Learned motion matching. *Acm transactions on graphics*, 39(4) (), 8 July, 53:53:1–53:53:12.

Holden, D., Komura, T. and Saito, J., 2017. Phase-Functioned Neural Networks for Character Control. *Acm transactions on graphics*, 36(4), 42:1–42:13.

Jankowski, J. and Hachet, M., 2013. A Survey of Interaction Techniques for Interactive 3D Environments. *Proceedings of the 34th Annual Conference of the European Association for Computer Graphics: State of the Art Reports (Eurographics 2013 - STARs)*. The Eurographics Association, pp.65–93.

Janocha, K. and Czarnecki, W.M., 2017. *On Loss Functions for Deep Neural Networks in Classification* [Online]. arXiv: `1702.05659 [cs]`.

Jennett, C., Cox, A.L., Cairns, P., Dhoparee, S., Epps, A., Tijs, T. and Walton, A., 2008. Measuring and defining the experience of immersion in games. *International journal of human-computer studies*, 66(9), pp.641–661.

Kallio, S., Kela, J. and Mäntyjärvi, J., 2003. Online gesture recognition system for mobile interaction. *Proceedings of the 2003 IEEE International Conference on Systems, Man and Cybernetics (SMC '03)*, 8 October 2003, Washington, DC, USA. Vol. 3. IEEE, pp.2070–2076.

Kandalaft, M.R., Didehbani, N., Krawczyk, D.C., Allen, T.T. and Chapman, S.B., 2013. Virtual Reality Social Cognition Training for Young Adults with High-Functioning Autism. *Journal of autism and developmental disorders*, 43(1), pp.34–44.

Kartsidis, P., Moustakas, N., Athanasiou, A., Astaras, A. and Bamidis, P.D., 2014. Comparative analysis of perceived psychometric characteristics: pilot study of the "Mercury" 6-degree of freedom robotic arm. *Proceedings of the 7th Electrical and Computer Engineering Student Conference (ECESCON 7)*, 11–13 April 2014, Thessaloniki, Greece, pp.179–181.

Keskin, C., Cemgil, A.T. and Akarun, L., 2011. DTW Based Clustering to Improve Hand Gesture Recognition. *Human Behavior Understanding*. Proceedings of the Second International Workshop on Human Behavior Understanding (HBU 2011), 16 November 2011, Amsterdam, The Netherlands. Springer, pp.72–81.

Kim, S., Park, G., Yim, S., Choi, S. and Choi, S., 2009. Gesture-recognizing hand-held interface with vibrotactile feedback for 3D interaction. *Ieee transactions on consumer electronics*, 55(3), pp.1169–1177.

Kingma, D.P. and Ba, J., 2015. Adam: A Method for Stochastic Optimization. *Proceedings of the 3rd International Conference on Learning Representations (ICLR 2015)*, 7–9 May 2015, San Diego, CA, USA.

Kingma, D.P. and Welling, M., 2014. *Auto-Encoding Variational Bayes* [Online]. 1 May. 1 May. arXiv: `1312.6114 [cs, stat]`.

Kolkmeier, J., Vroon, J. and Heylen, D., 2016. Interacting with Virtual Agents in Shared Space: Single and Joint Effects of Gaze and Proxemics. *Proceedings of the 16th International Conference on Intelligent Virtual Agents (IVA 2016)*, 20–23 September 2016, Los Angeles, CA, USA. Springer, 20 September, pp.1–14.

Kong, Y., Gao, S., Sun, B. and Fu, Y., 2018. Action Prediction from Videos via Memorizing Hard-to-Predict Samples. *Proceedings of the 32nd AAAI Conference on Artificial Intelligence (AAAI-18)*, 2–7 February 2018, New Orleans, LA, USA. AAAI Press, pp.7000–7007.

Kong, Y., Kit, D. and Fu, Y., 2014. A Discriminative Model with Multiple Temporal Scales for Action Prediction. In: D. Fleet, T. Pajdla, B. Schiele and T. Tuytelaars, eds. *Proceedings of the 13th European Conference on Computer Vision (ECCV 2014)*, 6–7 September 2014, Zurich, Switzerland. Cham: Springer International Publishing, pp.596–611.

Kotseruba, I. and Tsotsos, J.K., 2016. *A Review of 40 Years of Cognitive Architecture Research: Core Cognitive Abilities and Practical Applications* [Online]. arXiv: `1610.08602 [cs]`.

Kovar, L. and Gleicher, M., 2004. Automated extraction and parameterization of motions in large data sets. *Acm transactions on graphics*, 23(3), p.559.

Kovar, L., Gleicher, M. and Pighin, F., 2002. Motion Graphs. *Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '02)*, 23–26 July 2002, San Antonio, TX, USA. ACM, pp.473–482.

Krizhevsky, A., Sutskever, I. and Hinton, G.E., 2012. ImageNet Classification with Deep Convolutional Neural Networks. *Advances in Neural Information Processing Systems 25*. Curran Associates, Inc., pp.1097–1105.

Laird, J.E., Newell, A. and Rosenbloom, P.S., 1987. SOAR: An architecture for general intelligence. *Artificial intelligence*, 33(1), pp.1–64.

LeCun, Y., Haffner, P., Bottou, L. and Bengio, Y., 1999. Object Recognition with Gradient-Based Learning. *Shape, Contour and Grouping in Computer Vision*, Lecture Notes in Computer Science. Springer, Berlin, Heidelberg, pp.319–345.

Lee, J. and Lee, K.H., 2006. Precomputing avatar behavior from human motion data. *Graphical models*. Special Issue on SCA 2004, 68(2) (), 1 March, pp.158–174.

Lee, Y., Wampler, K., Bernstein, G., Popović, J. and Popović, Z., 2010. Motion Fields for Interactive Character Locomotion. *ACM SIGGRAPH Asia 2010 Papers (SIGGRAPH ASIA '10)*, 15–18 December 2010, Seoul, Republic of Korea. ACM, 138:1–138:8.

Leshno, M., Lin, V.Y., Pinkus, A. and Schocken, S., 1993. Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. *Neural networks*, 6(6), pp.861–867.

Liang, R.-H. and Ouhyoung, M., 1998. A real-time continuous gesture recognition system for sign language. *Proceedings of the 3rd IEEE International Conference on Automatic Face and Gesture Recognition (FG '98)*, 14–16 April 1998, Nara, Japan. IEEE, pp.558–567.

Ling, H.Y., Zinno, F., Cheng, G. and Van De Panne, M., 2020. Character controllers using motion VAEs. *Acm transactions on graphics*, 39(4), 40:40:1–40:40:12.

Llobera, J., Spanlang, B., Ruffini, G. and Slater, M., 2010. Proxemics with Multiple Dynamic Characters in an Immersive Virtual Environment. *Acm trans. appl. percept.*, 8(1), 3:1–3:12.

Lu, W., Tong, Z. and Chu, J., 2016. Dynamic Hand Gesture Recognition With Leap Motion Controller. *Ieee signal processing letters*, 23(9), pp.1188–1192.

Luo, Y.-S., Soeseno, J.H., Chen, T.P.-C. and Chen, W.-C., 2020. CARL: controllable agent with reinforcement learning for quadruped locomotion. *Acm transactions on graphics*, 39(4), 38:38:1–38:38:10.

Ma, S., Sigal, L. and Sclaroff, S., 2016. Learning Activity Progression in LSTMs for Activity Detection and Early Detection. *Proceedings of the 2016 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2016)*, 27–30 June 2016, Las Vegas, NV, USA. Las Vegas, NV, USA: IEEE, pp.1942–1950.

Mäntyjärvi, J., Kela, J., Korpipää, P. and Kallio, S., 2004. Enabling Fast and Effortless Customisation in Accelerometer Based Gesture Interaction. *Proceedings of the 3rd International Conference on Mobile and Ubiquitous Multimedia (MUM '04)*, 27–29 October 2004, College Park, MD, USA. ACM, pp.25–31.

Mantyla, V.-M., Mantyjarvi, J., Seppanen, T. and Tuulari, E., 2000. Hand gesture recognition of a mobile device user. *Proceedings of the 2000 IEEE International Conference on Multimedia and Expo (ICME)*, 30 July 2000–2 August 2004, New York City, NY, USA. Vol. 1. IEEE, pp.281–284.

McCulloch, W.S. and Pitts, W., 1943. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4), pp.115–133.

Mitra, S. and Acharya, T., 2007. Gesture Recognition: A Survey. *Ieee transactions on systems, man, and cybernetics, part c (applications and reviews)*, 37(3), pp.311–324.

Mizuguchi, M., Buchanan, J. and Calvert, T., 2001. Data driven motion transitions for interactive games. *Eurographics 2001 - Short Presentations*, 5–7 September 2001, Manchester, UK. Eurographics Association.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S. and Hassabis, D., 2015. Human-level control through deep reinforcement learning. *Nature*, 518(7540), pp.529–533.

Mohd Izani, A., Eshaq, A.R. and Norhan, N., 2003. Keyframe animation and moition capture for creating animation: a survey and perception from industry people. *Proceedings of the 11th Student Conference on Research and Development (SCOReD 2003)*, 25–26 August 2003, Putrajaya, Malaysia. IEEE, pp.154–159.

Molchanov, P., Yang, X., Gupta, S., Kim, K., Tyree, S. and Kautz, J., 2016. Online Detection and Classification of Dynamic Hand Gestures with Recurrent 3D Convolutional Neural Networks. *Proceedings of the 2016 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2016)*, 27–30 June 2016, Las Vegas, NV, USA. IEEE, pp.4207–4215.

Mori, M., MacDorman, K.F. and Kageki, N., 2012. The Uncanny Valley [From the Field]. *Ieee robotics automation magazine*, 19(2), pp.98–100.

Nair, V. and Hinton, G.E., 2010. Rectified Linear Units Improve Restricted Boltzmann Machines. *Proceedings of the 27th International Conference on Machine Learning (ICML '10)*, 21–24 June 2010, Haifa, Israel. Omnipress, pp.807–814.

Neverova, N., Wolf, C., Taylor, G.W. and Nebout, F., 2014. Multi-scale Deep Learning for Gesture Detection and Localization. *Workshops at the 13th European Conference on Computer Vision (ECCV 2014)*, 6–7 September 2014, Zurich, Switzerland. Springer, pp.474–490.

Noma, T., Zhao, L. and Badler, N.I., 2000. Design of a virtual human presenter. *Ieee computer graphics and applications*, 20(4) (), July, pp.79–85.

Nwankpa, C., Ijomah, W., Gachagan, A. and Marshall, S., 2018. *Activation Functions: Comparison of trends in Practice and Research for Deep Learning* [Online]. arXiv: `1811.03378 [cs]`.

Oord, A. van den, Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., Kalchbrenner, N., Senior, A. and Kavukcuoglu, K., 2016. *WaveNet: A Generative Model for Raw Audio* [Online]. arXiv: `1609.03499 [cs]`.

Perlin, K., 1995. Real time responsive animation with personality. *Ieee transactions on visualization and computer graphics*, 1(1) (), March, pp.5–15.

Rabiner, L.R. and Juang, B.-H., 1986. An introduction to hidden Markov models. *Ieee assp magazine*, 3(1), pp.4–16.

Raffa, G., Lee, J., Nachman, L. and Song, J., 2010. Don't slow me down: Bringing energy efficiency to continuous gesture recognition. *Proceedings of the 2010 International Symposium on Wearable Computers (ISWC)*, 10–13 October 2010, Seoul, Republic of Korea. IEEE, pp.1–8.

Rao, A.S. and Georgeff, M.P., 1991. Modeling Rational Agents within a BDI-Architecture. *Proceedings of the 2nd International Conference on Principles of Knowledge Representation and Reasoning (KR'91)*, 22–25 April 1991, Cambridge, MA, USA. Morgan Kaufmann, pp.473–484.

Reyes, M., Domínguez, G. and Escalera, S., 2011. Featureweighting in dynamic timewarping for gesture recognition in depth data. *Proceedings of the 2011 IEEE International Conference on Computer Vision Workshops (ICCV Workshops)*, 6–13 November 2011, Barcelona, Spain. IEEE, pp.1182–1188.

Rickel, J. and Johnson, W.L., 1999. Animated agents for procedural training in virtual reality: Perception, cognition, and motor control. *Applied artificial intelligence*, 13(4-5), pp.343–382.

Rosenbrock, H.H., 1960. An Automatic Method for Finding the Greatest or Least Value of a Function. *The computer journal*, 3(3), pp.175–184.

Rumelhart, D.E., Hinton, G.E. and Williams, R.J., 1986. Learning representations by back-propagating errors. *Nature*, 323(6088), pp.533–536.

Ryan, R.M., 1982. Control and information in the intrapersonal sphere: An extension of cognitive evaluation theory. *Journal of personality and social psychology*, 43(3), pp.450–461.

Sagar, M., 2015. BabyX. *Proceedings of the ACM SIGGRAPH 2015 Computer Animation Festival (SIGGRAPH '15)*, 9–13 August 2015, Los Angeles, CA, USA. ACM, pp.184–184.

Schlömer, T., Poppinga, B., Henze, N. and Boll, S., 2008. Gesture Recognition with a Wii Controller. *Proceedings of the 2nd International Conference on Tangible and Embedded Interaction (TEI '08)*, 18–20 February 2008, Bonn, Germany. ACM, pp.11–14.

Scott, D.W., 1992. *Multivariate Density Estimation: Theory, Practice, and Visualization*. New York: Wiley. 317pp.

Shoemake, K., 1985. Animating rotation with quaternion curves. *Proceedings of the 12th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '85)*, 22–26 July 1985, San Francisco, CA, USA. ACM, pp.245–254.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I. and Salakhutdinov, R., 2014. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1), pp.1929–1958.

Starke, S., Zhang, H., Komura, T. and Saito, J., 2019. Neural State Machine for Character-scene Interactions. *Acm transactions on graphics*, 38(6), 209:1–209:14.

Starke, S., Zhao, Y., Komura, T. and Zaman, K., 2020. Local motion phases for learning multi-contact character movements. *Acm transactions on graphics*, 39(4) (), 8 July, 54:54:1–54:54:13.

Syrdal, D.S., Dautenhahn, K., Koay, K.L., Walters, M.L. and Ho, W.C., 2013. Sharing Spaces, Sharing Lives – The Impact of Robot Mobility on User Perception of a Home Companion Robot. *Proceedings of the 5th International Conference on Social Robotics (ICSR 2013)*, 27–29 October 2013, Bristol, UK. Springer, pp.321–330.

Taubert, N., Löffler, M., Ludolph, N., Christensen, A., Endres, D. and Giese, M.A., 2013. A Virtual Reality Setup for Controllable, Stylized Real-time Interactions Between Humans and Avatars with Sparse Gaussian Process Dynamical Models. *Proceedings of the ACM Symposium on Applied Perception (SAP '13)*, 22–23 August 2013, Dublin, Ireland. ACM, pp.41–44.

Taylor, G.W., Hinton, G.E. and Roweis, S.T., 2007. Modeling Human Motion Using Binary Latent Variables. *Advances in Neural Information Processing Systems 19*. MIT Press, pp.1345–1352.

Treuille, A., Lee, Y. and Popović, Z., 2007. Near-optimal character animation with continuous control. *Acm transactions on graphics*, 36(3), 7:1–7:7.

Tsironi, E., Barros, P. and Wermter, S., 2016. Gesture Recognition with a Convolutional Long Short-Term Memory Recurrent Neural Network. *Proceedings of the 24th European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (ESANN 2016)*, 27–29 April 2016, Bruges, Belgium. Ciaco - i6doc.com, pp.213–218.

Vogt, D., Grehl, S., Berger, E., Amor, H.B. and Jung, B., 2014. A Data-Driven Method for Real-Time Character Animation in Human-Agent Interaction. *Proceedings of the 14th International Conference on Intelligent Virtual Agents (IVA 2014)*, 26–29 August 2014, Boston, MA, USA. Springer, 26 August, pp.463–476.

Wang, S.B., Quattoni, A., Morency, L.-P., Demirdjian, D. and Darrell, T., 2006. Hidden Conditional Random Fields for Gesture Recognition. *Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR '06)*, 17–22 June 2006, New York, NY, USA. Vol. 2. IEEE, pp.1521–1527.

Weissmann, J. and Salomon, R., 1999. Gesture recognition for virtual reality applications using data gloves and neural networks. *Proceedings of the International Joint Conference on Neural Networks (IJCNN '99)*, 10–16 July 1999, Washington, DC, USA. Vol. 3. IEEE, pp.2043–2046.

Werbos, P.J., 1974. *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*. PhD thesis. Harvard University.

Witkin, A. and Popovic, Z., 1995. Motion Warping. *Proceedings of the 22nd Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '95)*, 6–11 August 1995, Los Angeles, CA, USA. ACM, pp.105–108.

Witmer, B.G. and Singer, M.J., 1998. Measuring Presence in Virtual Environments: A Presence Questionnaire. *Presence: teleoperators and virtual environments*, 7(3), pp.225–240.

Wong, S.C., Gatt, A., Stamatescu, V. and McDonnell, M.D., 2016. Understanding Data Augmentation for Classification: When to Warp? *Proceedings of the 2016 International Conference on Digital Image Computing: Techniques and Applications (DICTA '16)*, 30 November–2 December 2016, Gold Coast, QLD, Australia. IEEE, pp.1–6.

Wu, D., Pigou, L., Kindermans, P.-J., Le, N.D.-H., Shao, L., Dambre, J. and Odobez, J.-M., 2016. Deep Dynamic Neural Networks for Multimodal Gesture Segmentation and Recognition. *Ieee transactions on pattern analysis and machine intelligence*, 38(8), pp.1583–1597.

Xu, D., 2006. A Neural Network Approach for Hand Gesture Recognition in Virtual Reality Driving Training System of SPG. *Proceedings of the 18th International Conference on Pattern Recognition (ICPR '06)*, 20–24 August 2006, Hong Kong, China. Vol. 3. IEEE, pp.519–522.

Yamato, J., Ohya, J. and Ishii, K., 1992. Recognizing human action in time-sequential images using hidden Markov model. *Proceedings of the 1992 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR '92)*, 15–18 June 1992, Champaign, IL, USA. IEEE, pp.379–385.

Yannakakis, G.N. and Togelius, J., 2018. AI Methods. *Artificial Intelligence and Games.* Cham: Springer International Publishing, pp.29–88.

Yin, Y. and Davis, R., 2014. Real-time continuous gesture recognition for natural human-computer interaction. *Proceedings of the 2014 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC '14)*, 28 July–1 August 2014, Melbourne, Australia, pp.113–120.

Yu, F. and Koltun, V., 2015. Multi-Scale Context Aggregation by Dilated Convolutions. *Proceedings of the 4rd International Conference on Learning Representations (ICLR 2016)*, 2–4 May 2016, San Juan, Puerto Rico. 23 November.

Zhang, H., Starke, S., Komura, T. and Saito, J., 2018. Mode-Adaptive Neural Networks for Quadruped Motion Control. *Acm transactions on graphics*, 37(4), 145:1–145:11.